

Series on Coding Theory and Cryptology – Vol. 7



Editors

I. Woungang

S. Misra

S. C. Misra

SELECTED TOPICS IN INFORMATION AND CODING THEORY

Copyright of the works in this Book is vested with World Scientific Publishing. This eBook is allowed for individual use only and may not be copied, further disseminated, or hosted on any other third party website or repository without the copyright holder's written permission.

**SELECTED TOPICS IN
INFORMATION AND
CODING THEORY**

Series on Coding Theory and Cryptology

Editors: Harald Niederreiter (*National University of Singapore, Singapore*) and
San Ling (*Nanyang Technological University, Singapore*)

Published

- Vol. 1 Basics of Contemporary Cryptography for IT Practitioners
 by *B. Ryabko and A. Fionov*

- Vol. 2 Codes for Error Detection
 by *T. Kløve*

- Vol. 3 Advances in Coding Theory and Cryptography
 eds. *T. Shaska et al.*

- Vol. 4 Coding and Cryptology
 eds. *Yongqing Li et al.*

- Vol. 5 Advances in Algebraic Geometry Codes
 eds. *E. Martínez-Moro, C. Munuera and D. Ruano*

- Vol. 6 Codes over Rings
 ed. *P. Solé*

- Vol. 7 Selected Topics in Information and Coding Theory
 eds. *I. Woungang, S. Misra and S. Chandra Misra*



SELECTED TOPICS IN INFORMATION AND CODING THEORY

Editors

I. Woungang

Ryerson University, Canada

S. Misra

Indian Institute of Technology, Kharagpur, India

S. C. Misra

State University of New York at Buffalo, USA

 **World Scientific**

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

SELECTED TOPICS IN INFORMATION AND CODING THEORY

Series on Coding Theory and Cryptology — Vol. 7

Copyright © 2010 by World Scientific Publishing Co. Pte. Ltd.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the Publisher.

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

ISBN-13 978-981-283-716-5

ISBN-10 981-283-716-7

Typeset by Stallion Press

Email: enquiries@stallionpress.com

Printed in Singapore.

Dedicated to

Isaac's wife: Clarisse and sons: Clyde, Lenny, Kylian

Subhas's daughter: Devarati

Sudip's son: Devadeep

This page intentionally left blank

PREFACE

Overview and Goals

Information and Coding Theory research and applications are undergoing rapid advancements. The last few years have witnessed rapid advancements in Information and Coding Theory research and applications. This book provides a comprehensive guide to selected topics, both ongoing and emerging, in Information and Coding Theory. Consisting of contributions from well known and high profile researchers in their respective specialties, topics that are covered include applications of coding theory to computational complexity, algebraic combinatorics in coding theory, codes construction and existence, source coding, channel capacity, network coding, and few other selected topics in Information and Coding Theory research.

The book has been prepared keeping in mind that it needs to prove itself to be a valuable resource dealing with both the important core and the specialized issues in Information and Coding Theory. We hope that it will be a valuable reference for students, instructors, researchers, engineers, and industry practitioners in these fields. All of the chapters are integrated in a manner that renders the book as a supplementary reference volume and/or textbook for use in both undergraduate and graduate courses on Information and Coding Theory. Each chapter is of an expository, but also of a scholarly, tutorial, or survey style, on a particular topic within the scope of Information and Coding Theory.

Organization and Features

The book is organized into 15 chapters, each chapter written by topical area experts. These chapters are grouped into four parts.

Part 1 is devoted to the applications of coding theory to computational complexity, and is composed of three chapters: Chaps. 1–3. Chapter 1 discusses several theoretical methods for analyzing the linear complexity and related complexity measures and proposes several classes of interesting sequences with high linear complexity. Chapter 2 focuses on the construction of high coding gain lattices with low decoding complexity from good codes in larger dimensions, and proposes a possible lattice construction with high coding gain using turbo codes and Low Density Parity Check codes. Chapter 3 is dedicated to the issues of cooperative communication in wireless relay networks. Various constructions of the distributed space-time block codes with low maximum-likelihood decidability are surveyed, and new upper bounds on the maximum rate of certain classes of single-symbol decodable distributed space-time block codes are proposed.

Part 2 focuses on methods of algebraic combinatorics in coding theory, and methods of codes construction and existence. It is composed of four chapters: Chaps. 4–7. Chapter 4 discusses in-depth the interplay of coding theory and algebraic combinatorics, focusing on the interaction of codes with combinatorial designs. Chapter 5 discusses ways and results in which to define, construct, prove theorems, and analyze codes from group rings in general, using both zero-divisor and units within a group ring. The codes derived are described as either zero-divisor or unit-derived codes. Chapter 6 is a continuation of the work initiated in Chapter 5, by presenting a new algebraic group ring-based method for constructing codes with no short cycles in the check matrix, and a general algebraic method for constructing Low Density Parity Check codes with no short cycles. Chapter 7 presents the construction of some well-known classes of algebraic block codes, and discusses recent generalizations of quasi-cyclic codes, as well as some algebraic and combinatorial methods of obtaining new codes from existing ones.

Part 3 centers on source coding, channel capacity, and network coding issues. It is composed of three chapters: Chaps. 8–10. Chapter 8 introduces a new approach to estimation, prediction and hypothesis testing for time series based on ideas of universal coding or universal data compression. Chapter 9 presents a subjective approach to network coding, which is concerned with the deterministic multicast encoding of cyclic networks. This topic is presented at a level of detail that is not found elsewhere in the literature. Chapter 10 addresses the problem of transmission of several distributed sources over a multiple access channel with side information

at the sources and the decoder, and proposes a joint source channel coding approach, which generalizes previous results available on the studied problem.

Part 4 addresses other selected topics in Information and Coding Theory, and is composed of five chapters; Chaps. 11–15. Chapter 11 presents a tutorial exposition of Low Density Parity Check codes. Chapter 12 focuses on some selected topics in the theory of variable length codes, including connections with codes for constrained channels and sources. Chapter 13 deals with decoding techniques and methods for finding the minimum distance of linear codes by means of Gröbner bases. Chapter 14 presents an overview of cooperative diversity, along with latest advances and open issues in this evolving field. In Chap. 15, algebraic coding theory is used as an alternative way to define secure cryptographic primitives.

We list below some of the important features of this book, which, we believe, would make it a valuable resource for our readers:

- This book is designed, in structure and content, to aid the learning process with the intention of making the book useful at all learning levels.
- Most of the chapters of the book are authored by prominent academicians/researchers, practitioners, in Information and Coding Theory that have been working with these topics for quite a few years now and have thorough understanding of the concepts.
- The authors of this book are distributed in a large number of countries and most of them are affiliated with institutions of worldwide repute. This gives this book an international flavor.
- Most of the chapters in this book have a distinct section providing direction for future research, which, particularly, targets researchers working in these areas. We believe that this section should provide insight to the researchers about some of the current research issues.
- The authors of each chapter have attempted to provide a comprehensive bibliography, which should greatly help the readers interested further to dig into the topics.
- Most of the chapters of this book have a separate section outlining thoughts for practitioners. We believe that this section in every chapter will be particularly useful for industry practitioners working directly with the practical aspects behind enabling these technologies in the field.
- All chapters, except one, provide a set of questions at the end that can help in assessing the understanding of the readers. In most chapters, solutions are provided to some of these questions.

- To make the book useful for pedagogical purposes, all chapters of the book have a corresponding set of presentation slides. The slides can be obtained as a supplementary resource by contacting the publisher, World Scientific, Singapore.

We have made attempts in all possible ways we could to make the different chapters of the book look as much coherent and synchronized as possible. However, it cannot be denied that due to the fact chapters were written by different authors, it was not fully possible to fully achieve this task. We believe that this is a limitation of most edited books of this sort.

Target Audience

The book is written by primarily targeting the student community. This includes the students of all levels — those getting introduced to these areas, those having an intermediate level of knowledge of the topics, and those who are already knowledgeable about many of the topics. To keep up with this goal, we have attempted to design the overall structure and content of the book in such a manner that makes it useful at all learning levels. To aid in the learning process, almost all chapters have a *set of questions* at the end of the chapter. Also, in order that teachers can use this book for classroom teaching, the book also comes with *presentation slides* and *sample solutions* to exercise questions, which are available as supplementary resources.

The secondary audience for this book is the research community, whether they are working in the academia or in the industry. To meet the specific needs to this audience group, certain chapters of the book provide directions for future research.

Finally, we have also taken into consideration the needs to those readers, typically from the industries, who have quest for getting insight into the practical significance of the topics, i.e. how the spectrum of knowledge and the ideas are relevant for real-life applications of coding and information theory.

Supplementary Resources

As mentioned earlier, the book comes with *presentation slides* for each chapter, which can be used for classroom instruction by teachers.

Teachers can contact the publisher, World Scientific, Singapore, to get access to these resources.

Acknowledgments

We are extremely thankful to the 25 authors of the 15 chapters of this book, who have worked very hard to bring this unique resource forward for help of the student, researcher, and practitioner community. The authors were very much interactive at all stages of preparation of the book from initial development of concept to finalization. We feel it is contextual to mention that as the individual chapters of this book are written by different authors, the responsibility of the contents of each of the chapters lies with the concerned authors.

We are also very much thankful to our colleagues in the World Scientific publishing and marketing teams, in particular, Ms. Kimberly Chua, Ms. Chelsea Chin, and Ms. Mei Kian, who tirelessly worked with us and guided us in the publication process. Special thanks also go to them for taking special interest in publishing this book, considering the current worldwide market needs for such a book.

Finally, we would like to thank our parents, Mr. J. Sime, Ms. C. Seupa, Prof. J.C. Misra, Ms. Shorasi Misra, our wives Clarisse, Satamita, and Sulagna, and our children Clyde, Lenny, Kylian, Babai, and Tultuli, for the continuous support and encouragement they offered during the course of this project.

*Dr. Isaac Woungang
Toronto, ON, Canada*

*Dr. Subhas Chandra Misra
Kanpur, UP, India*

*Dr. Sudip Misra
Kharagpur, WB, India*

This page intentionally left blank

CONTENTS

Preface	vii
Contributors	xv
Part 1: Applications of Coding Theory to Computational Complexity	1
Chapter 1: Linear Complexity and Related Complexity Measures.. <i>Arne Winterhof</i>	3
Chapter 2: Lattice and Construction of High Coding Gain Lattices from Codes	41
<i>Mohammad-Reza Sadeghi</i>	
Chapter 3: Distributed Space-Time Codes with Low ML Decoding Complexity	77
<i>G. Susinder Rajan and B. Sundar Rajan</i>	
Part 2: Methods of Algebraic Combinatorics in Coding Theory/Codes Construction and Existence	119
Chapter 4: Coding Theory and Algebraic Combinatorics	121
<i>Michael Huber</i>	
Chapter 5: Block Codes from Matrix and Group Rings	159
<i>Paul Hurley and Ted Hurley</i>	
Chapter 6: LDPC and Convolutional Codes from Matrix and Group Rings	195
<i>Paul Hurley and Ted Hurley</i>	

Chapter 7: Search for Good Linear Codes in the Class of Quasi-Cyclic and Related Codes.....	239
<i>Nuh Aydin and Tsvetan Asamov</i>	
Part 3: Source Coding/Channel Capacity/ Network Coding	287
Chapter 8: Applications of Universal Source Coding to Statistical Analysis of Time Series.....	289
<i>Boris Ryabko</i>	
Chapter 9: Introduction to Network Coding for Acyclic and Cyclic Networks.....	339
<i>Ángela I. Barbero and Øyvind Ytrehus</i>	
Chapter 10: Distributed Joint Source-Channel Coding on a Multiple Access Channel.....	423
<i>Vinod Sharma and R. Rajesh</i>	
Part 4: Other Selected Topics in Information and Coding Theory	469
Chapter 11: Low-Density Parity-Check Codes and the Related Performance Analysis Methods.....	471
<i>Xudong Ma</i>	
Chapter 12: Variable Length Codes and Finite Automata.....	505
<i>Marie-Pierre Béal, Jean Berstel, Brian H. Marcus, Dominique Perrin, Christophe Reutenauer and Paul H. Siegel</i>	
Chapter 13: Decoding and Finding the Minimum Distance with Gröbner Bases: History and New Insights.....	585
<i>Stanislav Bulygin and Ruud Pellikaan</i>	
Chapter 14: Cooperative Diversity Systems for Wireless Communication.....	623
<i>Murat Uysal and Muhammad Mehboob Fareed</i>	
Chapter 15: Public Key Cryptography and Coding Theory.....	663
<i>Pascal Véron</i>	

CONTRIBUTORS

Nuh Aydin

Department of Mathematics
Kenyon College, Gambier, OH 43022, USA
aydinn@kenyon.edu

Tsvetan Asamov

Department of Mathematics
Kenyon College, Gambier, OH 43022, USA
asamovt@kenyon.edu

Marie-Pierre Béal

Institut Gaspard-Monge (IGM)
Université Paris-Est
77454 Marne-la-Vallée Cedex 2, Paris, France
beal@univ-mlv.fr

Jean Berstel

Institut Gaspard-Monge (IGM), Université Paris-Est
77454 Marne-la-Vallée Cedex 2, Paris, France
berstel@univ-mlv.fr

Dominique Perrin

Institut Gaspard-Monge (IGM)
Université Paris-Est
77454 Marne-la-Vallée Cedex 2, Paris, France
dominique.perrin@esiee.fr

Brian H. Marcus

University of British Columbia
B.C., Vancouver, Canada
marcus@math.ubc.ca

Christophe Reutenauer

LaCIM, Université du Québec à Montréal
Montréal, Canada
reutenauer.christophe@uqam.ca

Paul H. Siegel

Department of Electrical and Computer Engineering
University of California at San Diego, San Diego, USA
psiegel@ucsd.edu

Boris Ryabko

Siberian State University of Telecommunications
and Informatics and Institute of Computational Technology
of Siberian Branch of Russian Academy of Science, Russia
boris@ryabko.net

Stanislav Bulygin

Department of Mathematics
University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
bulygin@mathematik.uni-kl.de

Ruud Pellikaan

Department of Mathematics and Computing Science
Eindhoven University of Technology, P.O. Box 513
NL-5600 MB, Eindhoven, The Netherlands
g.r.pellikaan@tue.nl

Michael Huber

Institut für Mathematik, Technische Universität Berlin,
Straße des 17. Juni 136, D-10623, Berlin, Germany
mhuber@math.TU-Berlin.DE

Murat Uysal

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, Ontario, Canada, N2L3G1
muysal@ece.uwaterloo.ca

Muhammad Mehboob Fareed

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, Ontario, Canada
mmfareed@ece.uwaterloo.ca

Angela I. Barbero

Department of Applied Mathematics
University of Valladolid
47011 Valladolid, Spain
angbar@wmatem.eis.uva.es

Oyvind Ytrehus

Department of Informatics
University of Bergen, N-5020 Bergen, Norway
oyvind@ii.uib.no

G. Susinder Rajan

Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore 560012, India
susinder@ece.iisc.ernet.in

B. Sundar Rajan

Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore 560012, India
bsrajan@ece.iisc.ernet.in

Mohammad-Reza Sadeghi

Faculty of Mathematics and Computer Science
Amirkabir University of Technology Hafez Ave.
Tehran, Iran
msadeghi@aut.ac.ir

Vinod Sharma

Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore 560012, India
vinod@ece.iisc.ernet.in

R. Rajesh

Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore 560012, India
rajesh@pal.ece.iisc.ernet.in

Paul Hurley

IBM Research
Zurich Research Laboratory, Switzerland
pah@zurich.ibm.com

Ted Hurley

Department of Mathematics
National University of Ireland
Galway, Ireland
ted.hurley@nuigalway.ie

Pascal Véron

Institut de Mathématiques de Toulon
Université du Sud Toulon-Var
Toulon, France
veron@univ-tln.fr

Xudong Ma

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo
Ontario N2L 3G1, Canada
x3ma@bbcr.uwaterloo.ca

Chapter 12

VARIABLE-LENGTH CODES AND FINITE AUTOMATA

MARIE-PIERRE BÉAL*, JEAN BERSTEL*,
BRIAN H. MARCUS†, DOMINIQUE PERRIN*,
CHRISTOPHE REUTENAUER‡
and PAUL H. SIEGEL¶

**Institut Gaspard-Monge (IGM), Université Paris-Est, France*

†University of British Columbia, Vancouver, Canada

‡LaCIM, Université du Québec à Montréal, Canada

¶Department of Electrical and Computer Engineering, UCSD, USA

The aim of this chapter is to present, in appropriate perspective, some selected topics in the theory of variable-length codes. One of the domains of applications is lossless data compression. The main aspects covered include optimal prefix codes and finite automata and transducers. These are a basic tool for encoding and decoding variable-length codes. Connections with codes for constrained channels and sources are developed in some detail. Generating series are used systematically for computing the parameters of encodings such as length and probability distributions. The chapter contains numerous examples and exercises with solutions.

12.1. Introduction

Variable-length codes occur frequently in the domain of data compression. Historically, they appeared at the beginning of modern information theory with the seminal work of Shannon. One of the first algorithmic results is the construction, by Huffman, of an optimal variable-length code for a given weight distribution. Although their role in data communication has been limited by their weak tolerance to faults, they are nonetheless commonly used in contexts where error handling is less critical or is treated by other methods.

Variable-length codes are strongly related to automata, and one of the aims of this chapter is to highlight connections between these domains. Automata are labeled graphs, and their use goes beyond the field of coding. Automata can be used to implement encoders and decoders, such as for compression codes, modulation codes, and convolutional error correcting codes.

The use of variable-length codes in data compression is widespread. Huffman's algorithm is still frequently used in various contexts, and under various forms, including in its adaptive version. In particular, Huffman codes are frequently used in the compression of motion pictures. In another setting, search trees are strongly related to ordered prefix codes, and optimal ordered prefix codes, as constructed later, correspond to optimal binary search trees.

Coding for constrained channels is required in the context of magnetic or optical recording. The constraints that occur can be represented by finite automata, and a coding method makes use of finite transducers. In this context, the constraints are defined by an automaton, which in turn is used to define a state-dependent encoder. Even if this encoder operates at fixed rate, it can also be considered as a memoryless encoder based on two variable-length codes with the same length distribution, that is, with the same number of words for each length.

Although convolutional error correcting codes are fixed-length codes, their analysis involves use of finite automata because encoders and decoders are described in terms of labeled graphs.

Specific properties of automata correspond to properties of variable-length codes. Typically, unambiguity in automata corresponds to unique decipherability.

Variable-length codes are also used for the representation of natural languages. They play a role, for instance, in phonetic transcription of languages and in the transformation from text to speech, or in speech recognition. We will mention examples in Sec. 12.10.

The mathematical relationship between codes and automata is very deep, as shown in early pioneering investigations by Schützenberger. He discovered and developed a new branch of algebra relating unique decipherability of codes with the theory of semigroups. There are still difficult open problems in the theory of variable-length codes. One of them is the commutative equivalence conjecture. It has practical significance in relation with optimal coding. We will discuss this and other open problems in Sec. 12.13.

The material covered by the chapter is as follows. We start with a few definitions and examples, and then address the problem of constructing optimal codes under various types of constraints. Typically, we consider alphabetic coding under cost conditions.

We study in detail prefix codes used for representing integers, such as Elias and Golomb codes.

Automata and transducers are introduced insofar as coding and decoding operations are concerned. These are applied to two special domains, namely coding with constraints on channels and constraints on sources. They are important in applications, which are described here.

Reversible codes, also called bifix codes, have both practical significance and deep mathematical properties, which we only sketch here.

The final section is concerned with synchronization. This is important in the context of error recovery, and we present very recent theoretical results such as the road coloring theorem.

The chapter is written at a level accessible to non-specialists. There are few formal proofs, but key algorithms are described in considerable detail and many illustrative examples are given. Exercises that reinforce and extend the material are given at the end of most sections, and sketches of solutions are provided. Some elementary questions and answers are also included.

12.2. Background

The topic of this chapter is an introduction to some syntactic and algorithmic problems of data transmission. In this sense, it is connected with three main fields:

- (1) coding and information theory;
- (2) automata and formal language theory;
- (3) algorithms.

In this section, we describe these connections, their historical roots and the notions to which they relate.

The relationship between codes and automata can be traced back to Shannon, who used labeled graphs to represent information sources. Later, the notion of *information lossless* machine was introduced as a model for reversible encoding [44]. These are the unambiguous transducers defined below. The term lossless has remained in common use with the notion of *lossless methods* for data compression. The main motivation for studying

variable-length codes is in data compression. In many coding systems, encoding by variable-length codes is used in connection with other coding components for error correction. The search for efficient data compression methods leads to algorithmic problems such as the design of optimal prefix codes under various criteria.

One of the main tools for encoding and decoding as presented here is finite automata and transducers. In this context, finite automata are particular cases of more general models of machines, such as pushdown automata that can use an auxiliary stack and, more generally, Turing machines that are used to define the notion of computability. The theory of automata has also developed independently of coding theory with motivations in algorithms on strings of symbols, in the theory of computation and also as a model for discrete processes.

The basic result of automata theory is *Kleene's theorem* asserting the equivalence between finite automata and regular expressions and providing algorithms to convert from automata to regular expressions and conversely. This conversion is actually used in the context of convolutional codes to compute the *path weight enumerator* (also called the *transfer function*) of a convolutional code [48].

Inside the family of finite automata, several subclasses have been studied, which correspond to various types of restrictions. An important one is the class of *aperiodic* automata, which contains the classes of local automata frequently used in connection with coding, in particular with sliding block encoders and decoders.

One of the possible extensions of automata theory is the use of multiplicities, which can be integers, real numbers, or elements of other semirings (see [12, 18]). The ordinary case corresponds to the Boolean semiring with just two elements 0, 1. This leads to a theory in which sets of strings are replaced by functions from strings to a semiring. This point of view has in particular the advantage of allowing the handling of *generating series* and gives a method, due to Schützenberger, to compute them. We will often use this method to compute generating series.

The well-known Huffman algorithm, described below, is the ancestor of a family of algorithms used in the field of information searching. Indeed, it can be used to build search trees as well as optimal prefix codes for source compression. The design and analysis of search algorithms is part of an important body of knowledge encompassing many different ideas and methods, including, for example, the theory of hashing functions (see [43]). Text processing algorithms find application in a variety of domains,

ranging from bioinformatics to the processing of large data sets such as those maintained by Internet search engines (see [46] for an introduction).

The topic of coding with constraints is related to symbolic dynamics, which is a field in its own right. Its aim is to describe dynamical systems and mappings between them. Codes for constrained channels are a particular case of these mappings (see [45]).

12.3. Thoughts for Practitioners

In this chapter, a wide variety of variable-length coding techniques are presented. We consider source and channel models that take into account the statistical properties of the source and the costs associated with transmission of channel symbols. Given these models, we define a measure of code optimality by which to evaluate the code design algorithms. The resulting families of variable-length codes are intended for use in a range of practical applications, including image and video coding, speech compression, magnetic and optical recording, data transmission, natural language representation, and tree search algorithms.

In practice, however, there are often system-related issues that are not explicitly reflected in the idealized source and channel models, and therefore are not taken into account by the code design algorithms. For example, there are often tradeoffs between the efficiency of the code and the complexity of its implementation in software or hardware. Encoding and decoding operations may be subject to latency constraints or a requirement for synchronous input–output processing. There is often a need for resilience against errors introduced by the channel, as well as robustness in the presence of variability in the source and channel characteristics. There may also be a system interface that dictates the incorporation of specific code properties or even a specific code. Finally, intellectual property concerns, such as the existence of patents on certain codes or coding methods, can play a role in practical code design. Such realities provide a challenge to the coding practitioner in applying the various design methods, as well as a stimulus for further research and innovation.

To illustrate some of these points, we examine two applications where variable-length coding has found pervasive use — data compression and digital magnetic recording.

Codes for data compression. In Sec. 12.6, we present the classical Huffman algorithm for designing an optimal prefix code for a memoryless source with specified symbol statistics and equal channel symbol costs. Codes produced

by this algorithm and its variants have been extensively employed in data compression applications. Practical system issues are often addressed during the code design process, or by using a modification of the standard design approach.

As will be described later in the chapter, for a binary channel alphabet, the Huffman algorithm builds a binary code tree from the bottom up by combining two nodes with the smallest probabilities. If there are multiple possibilities for selecting such pairs of nodes, all choices lead to codes with the same average codeword length. However, this is not true of the variance of the codeword lengths, and a large variance could have an impact on the implementation complexity if the code is incorporated into a data transmission system that calls for a constant transmission rate. This problem can be mitigated if the practitioner follows a simple rule for judiciously combining nodes during the generation of the code tree, resulting in a Huffman code with the smallest possible variance. There are also variants of the Huffman coding that help to control the maximum length of a codeword, a parameter that also may influence implementation complexity of the code.

Another practical consideration in applying Huffman codes may be the ease of representing the code tree. The class of *canonical* Huffman codes has a particularly succinct description: the codewords can be generated directly from a suitably ordered list of codeword lengths. Canonical codes are also very amenable to fast decoding and are of particular interest when the source alphabet is large. Fortunately, a code designed using the standard Huffman algorithm can be directly converted into a canonical code.

The practitioner may also encounter situations where the source statistics are not known in advance. In order to deal with this situation, one can use *adaptive* Huffman coding techniques. Application of adaptive coding, though, requires careful attention to a number of implementation related issues.

Huffman codes are generally not resilient to channel symbol errors. Nevertheless they have some inherent synchronization capabilities and, for certain length distributions, one can design synchronized Huffman codes. Typically, though, in order to ensure recovery within a reasonable time after a channel error, substantial modifications to the coding scheme are necessary.

Despite the availability of more efficient data compression methods, such as arithmetic coding, Huffman coding and its variants continue to play a role in many text and multimedia compression systems. They are

relatively effective, simple to implement, and, as just discussed, they offer some flexibility in coping with a variety of practical system issues. For further details, see, for example, [58].

Codes for digital magnetic recording. In Sec. 12.9, we consider the problem of encoding binary source sequences into binary channel sequences in which there are at least d and at most k 0's between consecutive 1's, for specified $0 \leq d < k$. This $[d, k]$ -constraint is used as a channel model in magnetic recording.

One simple approach to encoding a binary source sequence into the $[d, k]$ -constraint uses the idea of a *bit-stuffing* encoder. The bit-stuffing encoder generates a code sequence by inserting extra bits into the source sequence to prevent violations of the $[d, k]$ -constraint. It uses a counter to keep track of the number of consecutive 0's in the generated sequence. When the number reaches k , the encoder inserts a 1 followed by d 0's. Whenever the source bit is a 1, the encoder inserts d 0's. The decoder is correspondingly simple. It also keeps track of the number of consecutive 0's. Whenever the number reaches k , it removes the following $d + 1$ bits (the 1 and d 0's that had been inserted by the encoder). When the decoder encounters a 1, it removes the following d bits (the d 0's that had been inserted by the encoder). The bit-stuffing encoder can also be recast as a variable-length code, as shown for the special case $[d, k] = [2, 7]$ in Fig. 12.1. A source sequence can be uniquely parsed into a sequence of the source words shown, possibly followed by a prefix of a word. Each source word is then encoded into a corresponding channel word according to the table. Although the encoding and decoding operations are extremely simple conceptually, and the decoder resynchronizes with only a moderate delay following an erroneous channel bit, this code is not suitable for use in a disk drive because it does not have fixed encoding and decoding rate.

<i>source</i>	<i>channel</i>
1	001
01	0001
001	00001
0001	000001
00001	0000001
00000	00000001

Fig. 12.1. Variable-length $[2, 7]$ bit-stuffing code.

<i>source</i>	<i>channel</i>
10	0100
11	1000
000	000100
010	100100
011	001000
0010	00100100
0011	00001000

Fig. 12.2. The $[2, 7]$ Franaszek code.

In Sec. 12.9, we present another variable-length $[2, 7]$ code, the Franaszek code, whose encoder mapping is shown in Fig. 12.2. This encoder has a fixed encoding rate of $1/2$, since each source word is mapped to a codeword with twice its length. In fact, as shown in Sec. 12.9, this code can be implemented by a rate $1:2$, 6-state encoder that converts each source bit synchronously into two channel bits according to simple state-dependent rules. The decoder is also synchronous, and it decodes a pair of channel bits into a single source bit based upon the contents of an 8-bit window containing the channel bit pair along with the preceding pair and the next two upcoming pairs. This sliding-window decoding limits to four bits the propagation of decoding errors caused by an erroneous channel bit.

This code is also efficient, in the sense that the theoretical upper bound on the rate of a code for the $[2, 7]$ constraint is approximately 0.5172. Moreover, among $[2, k]$ -constraints, the $[2, 7]$ -constraint has the smallest k constraint that can support a rate $1/2$ code. This code was patented by IBM and was extensively used in its commercial disk drive products.

The construction of the Franaszek code involved certain choices along the way. For example, a different choice of the assignment of source words to channel codewords would potentially affect the implementation complexity as well as the worst-case decoder error propagation. Virtually all code design methodologies share this characteristic, and the practitioner has to exercise considerable technical insight in order to construct the best code for the particular situation.

For example, using the state-splitting algorithm discussed later in the chapter, one can construct another rate $1:2$ code for the $[2, 7]$ -constraint that requires only five encoder states; however, the resulting maximum error propagation of the decoder is increased to five bits. It is also possible

to construct fixed-rate encoders for the $[2,7]$ -constraint that have a higher code rate than the Franaszek code; however, the encoder and decoder implementation will very likely be more complex.

While some of the earliest $[d,k]$ -constrained codes designed for disk drives were standardized by the interface between the drive and the host computer drive controller, the encoding and decoding functions eventually migrated into the drive itself. This allowed coding practitioners to exercise their creativity and invent new codes to meet their particular system requirements. For further details about practical constrained code design, see, for example, [39, 47].

12.4. Definitions and Notation

We start with some definitions and notation on words and languages.

Some notation. Given a finite set A called the *alphabet*, each element of A is a *letter*, and a finite sequence of letters is called a *word*. The *length* of a word is the number of its letters. The length of a word w is denoted by $|w|$. The *empty word*, usually denoted by ε , is the unique word of length 0. The set of all words over the alphabet A is denoted by A^* . We denote by juxtaposition the *concatenation* of words. If w, x, y, z are words, and if $w = xy$, then x is a *prefix*, and y is a *suffix* of w . If $w = xyz$, then y is called a *factor* (or also a subword or a block) of w .

Given sets X and Y of words over some alphabet A , we denote by XY the set of all words xy , for x in X and y in Y . We write X^n for the n -fold product of X , with $X^0 = \{\varepsilon\}$. We denote by X^* the set of all words that are products of words in X , formally

$$X^* = \{\varepsilon\} \cup X \cup X^2 \cup \dots \cup X^n \cup \dots$$

If $X = \{x\}$, we write x^* for $\{x\}^*$. Thus, $x^* = \{x^n \mid n \geq 0\} = \{\varepsilon, x, x^2, x^3, \dots\}$. The operations of union, set product, and star (*) are used to describe sets of words by so-called *regular expressions*.

Generating series. Given a set of words X , the *generating series* of the lengths of the words of X is the series in the variable z defined by

$$f_X(z) = \sum_{x \in X} z^{|x|} = \sum_{n \geq 0} a_n z^n,$$

where a_n is the number of words in X of length n . It is easily checked that

$$f_{X \cup Y}(z) = f_X(z) + f_Y(z), \quad (12.1)$$

whenever X and Y are disjoint, and

$$f_{XY}(z) = f_X(z)f_Y(z), \quad (12.2)$$

when the product XY is *unambiguous*, that is whenever $xy = x'y'$ with $x, x' \in X$, $y, y' \in Y$ we have $x = x'$, $y = y'$. We will also make use of an extension (introduced later) of these series to the case where words are equipped with a cost.

Encoding. We start with a *source* alphabet B and a *channel* alphabet A . Consider a mapping γ that associates with each symbol b in B a non-empty word over the alphabet A . This mapping is extended to words over B by $\gamma(s_1 \cdots s_n) = \gamma(s_1) \cdots \gamma(s_n)$. We say that γ is an *encoding* if it is *uniquely decipherable* (UD) in the sense that

$$\gamma(w) = \gamma(w') \Rightarrow w = w'$$

for each pair of words w, w' . In this case, each $\gamma(b)$ for b in B is a *codeword*, and the set of all codewords is called a *variable-length code* or VLC for short. We will call this a *code* for short instead of the commonly used term UD code.

Every property of an encoding has a natural formulation in terms of a property of the associated code, and *vice versa*. We will generally not distinguish between codes and encodings.

Let C be a code. Since C^n and C^m are disjoint for $n \neq m$ and since the products C^n are unambiguous, we obtain from (12.1) and (12.2) the following fundamental equation:

$$f_{C^*}(z) = \sum_{n \geq 0} (f_C(z))^n = \frac{1}{1 - f_C(z)}. \quad (12.3)$$

Alphabetic encoding. Suppose now that both B and A are ordered. The *order* on the alphabet is extended to words lexicographically, that is, $u < v$ if either u is a proper prefix of v or $u = zaw$ and $v = zbw'$ for some words z, w, w' and letters a, b with $a < b$.

An encoding γ is said to be *ordered* or *alphabetic* if $b < b'$ implies $\gamma(b) < \gamma(b')$.

A set C of non-empty words over an alphabet A is a *prefix code* (*suffix code*) if no element of C is a proper prefix (suffix) of another one. An encoding γ of B is called a *prefix encoding* if the set $\gamma(B)$ is a prefix code.

Prefix codes are especially interesting because they are instantaneously decipherable in a left to right parsing.

Table 12.1. A binary ordered encoding of the five most frequent English words.

b	$\gamma(b)$
A	000
AND	001
OF	01
THE	10
TO	11

Example 12.1. The set B is composed of five elements in bijection with the five most common words in English, which are A, AND, OF, THE, and TO. An ordered prefix encoding γ of these words over the binary alphabet $\{0, 1\}$ is given in Table 12.1.

A common way to represent an encoding — one that is especially enlightening for prefix encodings — is by a rooted planar tree labeled in an appropriate way.

Assume the channel alphabet A has q symbols. The tree considered has nodes, which all have at most q children. The edge from a node to a child is labeled with one symbol of the channel alphabet A . If this alphabet is ordered, then the children are ordered accordingly from left to right. Some of the children may be missing.

Each path from the root to a node in the tree corresponds to a word over the channel alphabet, obtained by concatenating the labels on its edges. In this manner, a set of words is associated with a set of nodes in a tree, and conversely. If the set of words is a prefix code, then the set of nodes is the set of leaves of the tree.

Thus, a prefix encoding γ from a source alphabet B into words over a channel alphabet A is represented by a tree, and each leaf of the tree may in addition be labeled with the symbol b corresponding to the codeword $\gamma(b)$, which labels the path to this leaf. Figure 12.3 represents the tree associated with the ordered encoding γ of Table 12.1.

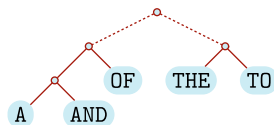


Fig. 12.3. The tree of the binary ordered encoding of the five most frequent English words.

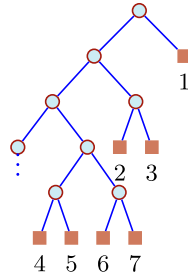


Fig. 12.4. The Elias code.

Example 12.2. The *Morse code* associates with each alphanumeric character a sequence of dots and dashes. For instance, *A* is encoded by “.-” and *J* is encoded by “.-.-.” Provided each codeword is terminated with an additional symbol (usually a space, called a “pause”), the Morse code becomes a prefix code.

Example 12.3. There are many representations of integers. The unary representation of an integer n is composed of a sequence of n symbols 1. The usual binary representation of positive integers is exponentially more succinct than the unary representation, and thus is preferable for efficiency. However, it is not adapted to representation of sequences of integers, since it is not UD: for instance, 11010 may represent the number 26, or the sequence 6, 2, or the sequence 1, 2, 2. The *Elias code* [19] maps a positive integer into a word composed of its binary representation preceded by a number of zeros equal to the length of this representation minus one. For instance, the Elias encoding of 26 is 000011010. It is easily seen that the set of Elias encodings of positive integers is a prefix code. The corresponding tree is given in Fig. 12.4.

Example 12.4. There exist codes, even quite simple ones, which are neither prefix nor suffix. This is the case of the encoding of a, b , and c by 00, 10, and 100. To see that this is indeed UD, one considers the occurrences of 1. If the number of 0’s following a 1 is even, this block is decoded as $ca \cdots a$; otherwise, as $ba \cdots a$.

12.5. Basic Properties of Codes

We start with stating a basic numerical inequality on codes. It gives a restriction on the distribution of lengths of codewords.

Kraft–McMillan inequality. For any code C over an alphabet A with k letters, one has the inequality, called the *Kraft–McMillan inequality* (see, for instance, [5])

$$\sum_{c \in C} k^{-|c|} \leq 1. \quad (12.4)$$

We prove the inequality below. Before that, we note that (12.4) is easy to prove for a finite prefix code. Indeed, the inequality above can also be written as

$$\sum_i a_i k^{-i} \leq 1,$$

where a_i is the number of code words in C of length i . Multiply both sides by k^n , where n is the maximal length of codewords. One gets $\sum_i a_i k^{n-i} \leq k^n$. The left-hand side counts the number of words of length n that have a prefix in C , and the inequality expresses the fact that each word of length n has at most one prefix in C .

For general codes, the inequality (12.4) can be proved as follows. Consider the *generating series* of the lengths of the words of the code C

$$f_C(z) = \sum_{c \in C} z^{|c|} = \sum_{n \geq 0} a_n z^n,$$

where a_n is the number of codewords of length n . Then, since C is a code, we have using (12.3):

$$f_{C^*}(z) = \frac{1}{1 - f_C(z)}.$$

Set $f_{C^*}(z) = \sum_{n \geq 0} b_n z^n$. Since C^* is a subset of A^* , one has $b_n \leq k^n$. Thus, the radius of convergence of $f_{C^*}(z)$ is at least equal to $1/k$. Since $f_C(r)$ is increasing for real positive r and $f_C(0) = 0$, the radius of convergence of $f_{C^*}(z)$ is precisely the positive real number r such that $f_C(r) = 1$. Thus, $f_C(1/k) \leq 1$. This proves the Kraft–McMillan inequality.

There is a converse statement for the Kraft–McMillan inequality: For any sequence ℓ_1, \dots, ℓ_n of positive integers such that $\sum_i k^{-\ell_i} \leq 1$, there exists a prefix code $C = \{c_1, \dots, c_n\}$ over A such that $|c_i| = \ell_i$.

This can be proved by induction on n as follows. It is clearly true for $n = 1$. Suppose that $n > 1$ and consider ℓ_1, \dots, ℓ_n satisfying $\sum_{i=1}^n k^{-\ell_i} \leq 1$. Since also $\sum_{i=1}^{n-1} k^{-\ell_i} \leq 1$, there exists by induction hypothesis a prefix code

$C = \{c_1, \dots, c_{n-1}\}$ such that $|c_i| = \ell_i$ for $1 \leq i \leq n-1$. We multiply both sides of the inequality $\sum_{i=1}^{n-1} k^{-\ell_i} \leq 1$ by k^{ℓ_n} , and we obtain

$$\sum_{i=1}^{n-1} k^{\ell_n - \ell_i} \leq k^{\ell_n} - 1. \quad (12.5)$$

Each of the terms $k^{\ell_n - \ell_i}$ of the left-hand side of (12.5) counts the number of words of length $\ell_n - \ell_i$, and can be viewed as counting the number of words of length ℓ_n with fixed prefix c_i of length ℓ_i . Since the code C is prefix, the sets of words of length ℓ_n with fixed prefix c_i are pairwise disjoint; hence, the left-hand side of (12.5) is the number of words of length ℓ_n on the alphabet A , which have a prefix in C . Thus, (12.5) implies that there exists a word c_n of length ℓ_n over the alphabet A , which does not have a prefix in C . The set $\{c_1, \dots, c_n\}$ is then a prefix code. This proves the property.

Entropy. Consider a source alphabet B . We associate with each symbol $b \in B$ a weight which we denote by $\text{weight}(b)$. For now, we assume that B is finite. The symbol weights are often *normalized* to sum to 1, in which case they can be interpreted as probabilities. The *entropy* of the source $B = \{b_1, \dots, b_n\}$ with probabilities $p_i = \text{weight}(b_i)$ is the number

$$H = - \sum_{i=1}^n p_i \log p_i,$$

where \log is the logarithm to base 2. Actually, this expression defines what is called the entropy of order 1. The same expression defines the entropy of order k , when the p_i 's are the probabilities of the blocks of length k and n is replaced by n^k .

Channel. In the context of encoding the symbols of a source alphabet B , we consider a channel alphabet A with a *cost*, denoted $\text{cost}(a)$, associated with each channel symbol $a \in A$. The cost of a symbol is a positive integer. The symbol costs allow us to consider the case where the channel symbols have non-uniform lengths, and the cost of each symbol can be interpreted as the time required to send the symbol. A classic example is the alphabet composed of two symbols $\{., -\}$, referred to as *dot* and *dash*, with costs 1 and 2, respectively. This alphabet is sometimes referred to as the *telegraph channel*.

The *channel capacity* is $\log 1/\rho$ where ρ is the real positive root of

$$\sum_{a \in A} z^{\text{cost}(a)} = 1.$$

In the case of an alphabet with k symbols, each having cost equal to 1, this reduces to $\rho = 1/k$. In the case of the telegraph channel, the capacity is the positive root of $\rho + \rho^2 = 1$, which is $\rho \approx 0.618$.

The cost of a word w over the alphabet A is denoted by $\text{cost}(w)$. It is by definition the sum of the costs of the letters composing it. Thus,

$$\text{cost}(a_1 \cdots a_n) = \text{cost}(a_1) + \cdots + \text{cost}(a_n).$$

We extend the notation of generating series as follows. For a set of words, X , denote by

$$f_X(z) = \sum_{x \in X} z^{\text{cost}(x)}$$

the *generating series of the costs* of the elements of X . For convenience, the cost function is omitted in the notation. Note that if the cost function assigns to each word its length, the generating series of costs reduces to the generating series of lengths considered earlier. Equations (12.1)–(12.3) hold for general cost functions.

For a code C over the alphabet A , the following inequality holds, which is a generalization of the Kraft–McMillan inequality (12.4):

$$\sum_{c \in C} \rho^{\text{cost}(c)} \leq 1. \quad (12.6)$$

The proof follows the same argument as for the Kraft–McMillan inequality using the generating series of the costs of the words of C

$$f_C(z) = \sum_{c \in C} z^{\text{cost}(c)} = \sum_{n \geq 0} a_n z^n,$$

where a_n is the number of words in C of cost equal to n . Similarly,

$$f_{A^*}(z) = \sum_{w \in A^*} z^{\text{cost}(w)} = \sum_{n \geq 0} b_n z^n,$$

where b_n is the number of words in A^* of cost equal to n . Now

$$f_{A^*}(z) = \frac{1}{1 - f_A(z)},$$

and the radius of convergence of $f_{A^*}(z)$ is the number ρ , which is the real positive root of $\sum_{a \in A} z^{\text{cost}(a)} = 1$.

Optimal encoding. Consider an encoding γ , which associates with each symbol b in B a word $\gamma(b)$ over the alphabet A . The *weighted cost* is

$$W(\gamma) = \sum_{b \in B} \text{weight}(b) \text{cost}(\gamma(b)).$$

When the weights are probabilities, Shannon's fundamental theorem on discrete noiseless channels [62] implies a lower bound on the weighted cost,

$$W(\gamma) \geq \frac{H}{\log 1/\rho}.$$

To show this, we set $B = \{b_1, \dots, b_n\}$, $p_i = \text{weight}(b_i)$, and $q_i = \rho^{\text{cost}(\gamma(b_i))}$. We can then write $(\log \rho)W(\gamma) = \sum p_i \log q_i$. Invoking the well-known inequality $\ln x \leq x - 1$, where \ln denotes the natural logarithm, and applying (12.6), we find

$$\begin{aligned} (\log \rho)W(\gamma) - \sum p_i \log p_i &= \sum p_i \log \frac{q_i}{p_i} = (\log e) \sum p_i \ln \frac{q_i}{p_i} \\ &\leq (\log e) \sum p_i \left(\frac{q_i}{p_i} - 1 \right) \\ &\leq (\log e) \left(\sum q_i - 1 \right) \leq 0, \end{aligned}$$

from which the bound follows.

The *optimal encoding problem* is the problem of finding, for given sets B and A with associated weight and cost functions, an encoding γ such that $W(\gamma)$ is minimal.

The *optimal prefix encoding problem* is the problem of finding an optimal prefix encoding. Most research on optimal coding has been devoted to this second problem, both because of its practical interest and because of the conjecture that an optimal encoding can always be chosen to be prefix (see [41] and also the discussion below).

There is another situation that will be considered below, where the alphabets A and B are ordered and the encoding is required to be ordered.

In the case of *equal letter costs*, that is, where all $\text{cost}(a)$ are equal, the cost of a letter may be assumed to be 1. The cost of a codeword $\gamma(b)$ is then merely the length $|\gamma(b)|$. In this case, the weighted cost is called the *average length* of codewords. An optimal encoding can always be chosen to be prefix in view of the Kraft–McMillan inequality, as mentioned above.

Commutative equivalence. In the case of *unequal letter costs*, the answer to the problem of finding a prefix encoding, which has the same weighted cost as an optimal encoding, is not known. This is related to an important conjecture, which we now formulate. Two codes C and D are *commutatively equivalent* if there is a one-to-one correspondence between C and D such that any two words in correspondence have the same number of occurrences of each letter (that is, they are anagrams). Observe that the encodings corresponding to commutatively equivalent codes have the same weight, and therefore one is optimal if the other is.

Example 12.5. The code $C = \{00, 10, 100\}$ seen in Example 12.4 is neither prefix nor suffix. It is commutatively equivalent to the prefix code $D = \{00, 01, 100\}$ and to the suffix code $D' = \{00, 10, 001\}$.

It is conjectured that any finite *maximal* code (that is, a code that is not strictly contained in another code) is commutatively equivalent to a prefix code. This would imply that, in the case of maximal codes, the optimal encoding can be obtained with a prefix code. For a discussion, see [10, 13]. The conjecture is known to be false if the code is not maximal. A counter-example has been given by Shor [63]. Note that if equality holds in the Kraft–McMillan inequality (12.4), then the code must be maximal. Conversely, it can be shown (see, [11]) that for a finite maximal code, (12.4) is an equality.

12.6. Optimal Prefix Codes

In this section, we describe methods used to obtain optimal prefix codes under various constraints, such as equal or unequal letter costs, as well as equal or unequal letter weights, and finally encodings which are alphabetic or not. The different cases are summarized in Fig. 12.5, where the vertices are associated with the inventors of some corresponding algorithm. For instance, vertex 3 denotes the problem of finding an optimal alphabetic tree in the case of unequal weights and unequal letter costs, and it is solved by Itai's algorithm described later. All these algorithms, except Karp's algorithm, have polynomial running time. We consider first the two cases (vertices 4 and 1 in Fig. 12.5) of unequal weights without the constraint on the encoding to be alphabetic.

Unequal letter costs. The computational complexity of the optimal prefix encoding problem in the case of unequal letter costs (vertex 4 in Fig. 12.5) is

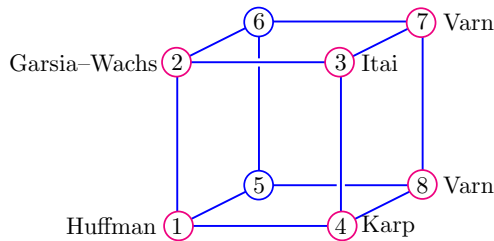


Fig. 12.5. Various hypotheses. Front plane (1, 2, 3, 4): unequal weights. Right plane (3, 4, 7, 8): unequal letter costs. Top plane (2, 3, 6, 7): alphabetic encodings. The two unnamed vertices 5, 6 are easy special cases.

still unknown, in the sense that neither polynomial time algorithm is known for this, nor is it known whether the corresponding recognition problem (is there a code of cost $\leq m$?) is NP-complete. It has been shown to be reducible to an integer programming problem by Karp [41].

We explain how an optimal prefix code can be found by solving an integer programming problem. Let ρ be the positive real number such that $f_A(\rho) = 1$. Thus, $\log 1/\rho$ is the channel capacity. Recall that, for any code C , one has

$$f_C(\rho) \leq 1. \quad (12.7)$$

However, given a series $f(z)$, the inequality $f(\rho) \leq 1$ is not sufficient to imply the existence of a prefix code C such that $f(z) = f_C(z)$. For example, if the alphabet A has a single letter of cost 2, then $f_A(z) = z^2$, and so $\rho = 1$. The polynomial $f(z) = z$ satisfies $f(\rho) = 1$; however, there can be no codeword of cost 1.

Despite this fact, the existence of a code C with prescribed generating series of costs can be formulated in terms of solutions for a system of linear equations, as we describe now.

Let C be a prefix code over the channel alphabet A , with source alphabet B equipped with weights denoted $\text{weight}(b)$ for $b \in B$, and let P be the set of prefixes of words in C which do not belong to C . Set

$$f_A(z) = \sum_{i \geq 1} a_i z^i, \quad f_C(z) = \sum_{i \geq 1} c_i z^i, \quad f_P(z) = \sum_{i \geq 1} p_i z^i.$$

Here a_i is the number of channel symbols of cost i , c_i is the number of codewords of cost i , and p_i is the number of words in P of cost i . The

following equality holds between the sets C, P , and A (see Exercise 12.6.2).

$$PA \cup \{\varepsilon\} = P \cup C.$$

Since the unions are disjoint, it follows that

$$\begin{aligned} c_1 + p_1 &= a_1, \\ c_2 + p_2 &= p_1 a_1 + a_2, \\ &\vdots \\ c_n + p_n &= p_{n-1} a_1 + \cdots p_1 a_{n-1} + a_n, \\ &\vdots \end{aligned} \tag{12.8}$$

Conversely, if c_i, a_i are non-negative integers satisfying these equations, there is a prefix code C such that $f_C(z) = \sum_{i \geq 1} c_i z^i$.

Thus, an optimal prefix code can be found by solving the problem of finding non-negative integers u_b for $b \in B$, which will be the costs of the codewords, and integers c_i, p_i for $i \geq 1$, which minimize the linear form $\sum_b u_b \text{weight}(b)$ such that Eq. (12.8) hold and with c_i equal to the number of b such that $u_b = i$.

There have been many approaches to partial solutions of the optimal prefix encoding problem [25, 49]. The most recent one is a polynomial time approximation scheme, which has been given in [29]. This means that, given $\epsilon > 0$, there exists a polynomial time algorithm computing a solution with weighted cost $(1 + \epsilon)W$, where W is the optimal weighted cost.

Equal letter costs. The case of equal letter costs (vertex 1 in Fig. 12.5) is solved by the well-known *Huffman algorithm* [37]. The principle of this algorithm in the binary case is as follows. Select two symbols b_1, b_2 in B with lowest weights, replace them by a fresh symbol b with weight $\text{weight}(b) = \text{weight}(b_1) + \text{weight}(b_2)$, and associate with b a node with children labeled b_1 and b_2 . Then iterate the process. The result is a binary tree corresponding to an optimal prefix code. The complexity of the algorithm is $O(n \log n)$, or $O(n)$ if the weights are available in increasing order. The case where all weights are equal (vertex 5 in Fig. 12.5) is an easy special case.

Example 12.6. Consider the alphabets $B = \{a, b, c, d, e, f\}$ and $A = \{0, 1\}$, and the weights given in the following table:

	a	b	c	d	e	f
weight	2	2	3	3	3	5

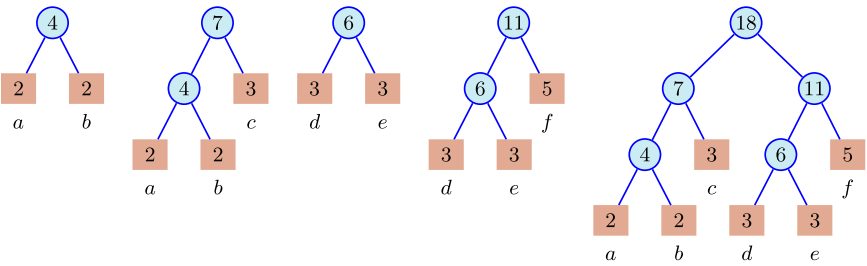


Fig. 12.6. Computing an optimal Huffman encoding by combining trees.

The steps of the algorithm are presented below:

$a\ b\ c\ d\ e\ f$	$ab\ c\ d\ e\ f$	$ab\ c\ de\ f$	$(ab)c\ de\ f$	$(ab)c\ (de)f$	$((ab)c)((de)f)$
2 2 3 3 3 5	4 3 3 3 5	4 3 6 5	7 6 5	7 11	18

The corresponding trees are given in Fig. 12.6.

Alphabetic coding. We suppose that both the source and the channel alphabets are ordered (these are vertices 2, 3, 6, 7 in Fig. 12.5). Recall that an encoding γ is said to be *ordered* or *alphabetic* if $b < b' \Rightarrow \gamma(b) < \gamma(b')$. The *optimal alphabetic prefix encoding problem* is the problem of finding an optimal ordered prefix encoding.

Alphabetic encoding is motivated by searching problems. Indeed, a prefix code can be used as a searching procedure to retrieve an element of an ordered set. Each node of the associated tree corresponds to a query, and the answer to this query determines the subtree in which to continue the search.

Example 12.7. In the binary tree of Figure 12.3, one looks for an occurrence of an English word. The query associated with the root can be the comparison of the first letter of the word to the letter T.

In contrast to the non-alphabetic case, there exist polynomial-time solutions to the optimal alphabetic prefix encoding problem. It has been considered mainly in the case where the channel alphabet is binary. Again, there is a distinction between equal letter costs and unequal letter costs.

Example 12.8. Consider the alphabet $B = \{a, b, c\}$, with $\text{weight}(a) = \text{weight}(c) = 1$ and $\text{weight}(b) = 4$. Figure 12.7(a) shows an optimum tree

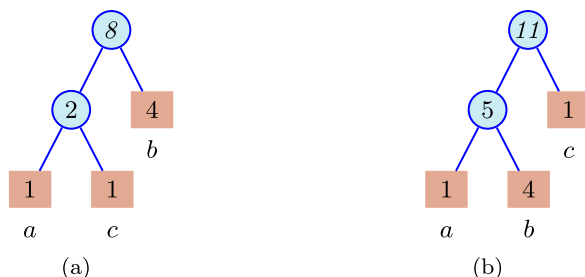


Fig. 12.7. Two trees for the given weights. Optimum coding tree (a) has weighted cost 8, it is optimal but not ordered. Optimum ordered tree (b) is ordered and has weighted cost 11.

for these weights, and Fig. 12.7(b) an optimum ordered tree. This example shows that Huffman's algorithm does not give the optimal ordered tree.

We consider now the case of equal letter costs, represented by vertex 2 in Fig. 12.5. Let $B = \{b_1, \dots, b_n\}$ be an ordered alphabet with n letters, and let p_i be the weight of letter b_i . There is a simple algorithm for computing an optimal ordered tree based on dynamic programming. It runs in time $O(n^3)$ and can be improved to run in time $O(n^2)$ (see [43]).

We present a more sophisticated algorithm due to Garsia and Wachs [23]. The intuitive idea of the algorithm is to use a variant of Huffman's algorithm by grouping together pairs of elements with minimal weight, which are consecutive in the ordering. The algorithm can be implemented to run in time $O(n \log n)$.

Example 12.9. Consider the following weights for an alphabet of five letters:

	a	b	c	d	e
weight	25	20	12	10	14

The algorithm is composed of three parts. In the first part, called the *combination* part, one starts with the sequence of weights

$$p = (p_1, \dots, p_n)$$

and constructs an optimal binary tree T' for a permutation $b_{\sigma(1)}, \dots, b_{\sigma(n)}$ of the alphabet. The leaves, from left to right, have weights

$$p_{\sigma(1)}, \dots, p_{\sigma(n)}.$$

In general, this permutation is not the identity; hence, the tree is not ordered, see Fig. 12.8(a).

The second part, the *level assignment*, consists of computing the levels of the leaves. In the last part, called the *recombination* part, one constructs a tree T , which has the weights p_1, \dots, p_n associated with its leaves from left to right, and where each leaf with weight p_i appears at the same level as in the previous tree T' . This tree is ordered, see Fig. 12.8(b).

Since the leaves have the same level in T and in T' , the corresponding codewords have the same length, and therefore the trees T and T' have the same weighted cost. Thus, T is an optimal ordered tree.

We now give the details of the algorithm and illustrate it with this specific example. For ease of description, it is convenient to introduce some terminology. A sequence (p_1, \dots, p_k) of numbers is *2-descending* if $p_i > p_{i+2}$ for $1 \leq i \leq k-2$. Clearly a sequence is 2-descending if and only if the sequence of “two-sums” $(p_1 + p_2, \dots, p_{k-1} + p_k)$ is strictly decreasing.

Let $p = (p_1, \dots, p_n)$ be a sequence of (positive) weights. The *left minimal pair* or simply *minimal pair* of p is the pair (p_{k-1}, p_k) , where (p_1, \dots, p_k) is the longest 2-descending chain that is a prefix of p . The index k is the *position* of the pair. In other words, k is the integer such that

$$p_{i-1} > p_{i+1} \quad (1 < i < k) \quad \text{and} \quad p_{k-1} \leq p_{k+1}$$

with the convention that $p_0 = p_{n+1} = \infty$. The *target* is the index j with $1 \leq j < k$ such that

$$p_{j-1} \geq p_{k-1} + p_k > p_j, \dots, p_k.$$

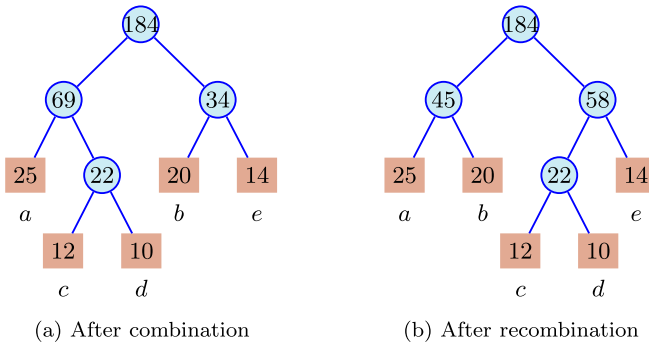


Fig. 12.8. The two steps of the algorithm: (a) the unordered tree obtained in the combination phase, and (b) the final ordered tree, obtained by recombination. Both have weighted cost 184.

Example 12.10. For $(14, 15, 10, 11, 12, 6, 8, 4)$, the minimal pair is $(10, 11)$, the target is 1, whereas for the sequence $(28, 8, 15, 20, 7, 5)$, the minimal pair is $(8, 15)$ and the target is 2.

The three phases of the algorithm work as follows. Let (p_1, \dots, p_n) be a sequence of weights.

Combination. Associate with each weight a tree composed of a single leaf. Repeat the following steps as long as the sequence of weights has more than one element:

- (i) Compute the *left minimal pair* (p_{k-1}, p_k) .
- (ii) Compute the *target* j .
- (iii) Remove the weights p_{k-1} and p_k .
- (iv) Insert $p_{k-1} + p_k$ between p_{j-1} and p_j .
- (v) Associate with $p_{k-1} + p_k$ a new tree with weight $p_{k-1} + p_k$, and which has a left (right) subtree the corresponding tree for p_{k-1} (for p_k).

Level assignment. Compute, for each letter b in B , the level of its leaf in the tree T' .

Recombination. Construct an ordered tree T in which the leaves of the letters have the levels computed by the level assignment.

Example 12.11. Consider again the following weights for an alphabet of five letters:

	a	b	c	d	e
weight	25	20	12	10	14

The initial sequence of trees is given in Fig. 12.9. The left minimal pair is 12, 10; its target is 2; hence, the leaves for c and d are combined into a tree which is inserted just to the right of the first tree, as shown in the left-hand side of Fig. 12.10. Now the minimal pair is $(20, 14)$ (there is an infinite weight at the right end); hence, the leaves for letters b and e are combined, and inserted at the beginning. The resulting sequence of trees is shown in the right-hand side of Fig. 12.10.

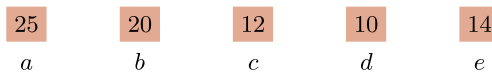


Fig. 12.9. The initial sequence of trees.



Fig. 12.10. The next two steps.

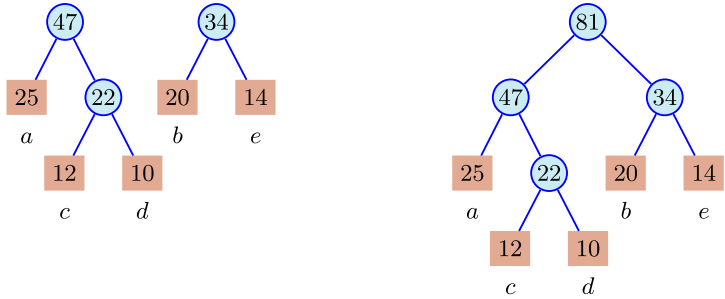


Fig. 12.11. The two last steps of the combination part.

Next, the last two trees are combined and inserted at the beginning as shown in the left-hand side of Fig. 12.11, and finally, the two remaining trees are combined, yielding the tree shown in the right-hand side of the figure.

The tree T' obtained at the end of the first phase is not ordered. The prescribed levels for the letters of the example are as follows:

	a	b	c	d	e
level	2	2	3	3	2

The optimal ordered tree with these levels is given by recombination. It is the tree given in Fig. 12.8(b).

For a correctness proof, see [42, 43]. The time bound is given in [43]. The Garsia–Wachs algorithm is simpler than a previous algorithm due to Hu and Tucker [35], which was also described in the first edition of Knuth’s book, preceding [43]. For a proof and a detailed description of the Hu–Tucker algorithm and variations, see [34, 36].

Alphabetic coding with unequal costs. This is the most general case for alphabetic encoding (vertex 3 in Fig. 12.5). There is a dynamic programming algorithm due to Itai [40], which computes an optimal solution in polynomial time.

Given a source alphabet $B = \{1, \dots, n\}$ with n symbols, and weights $\text{weight}(1), \dots, \text{weight}(n)$, one looks for an optimal alphabetic encoding γ on the ordered channel alphabet A with costs $\text{cost}(a)$, for a in A . The weighted cost is $\sum_{i=1}^n \text{weight}(i) \text{cost}(\gamma(i))$. For convenience, the first (resp. the last) letter in A is denoted by α (resp. by ω). We also write $a + 1$ for the letter following a in the order on A .

Define $W_{a,b}[i, j]$ as the minimal weight of an alphabetic encoding for the symbols k with $i \leq k \leq j$, using codewords for which the initial symbol x satisfies $a \leq x \leq b$.

The following equations provide a method to compute the minimal weight $W_{\alpha, \omega}[1, n]$. First, for $a < b$, $i < j$,

$$W_{a,b}[i, j] = \min\{W_{a+1,b}[i, j], V_{a,b}[i, j], W_{a,a}[i, j]\}, \quad (12.9)$$

where

$$V_{a,b}[i, j] = \min_{i \leq k < j} (W_{a,a}[i, k] + W_{a+1,b}[k+1, j]).$$

This formula expresses the fact that either the first codeword does not start with the letter a , or it does, and the set of codewords starting with a encodes the interval $[i, k]$ for some $k < j$, or finally all codewords start with a . Next, for $i < j$,

$$W_{a,a}[i, j] = \text{cost}(a) \left(\sum_{k=i}^j \text{weight}(k) \right) + \min_{\substack{i \leq k < j \\ \alpha \leq x < \omega}} \{W_{x,x}[i, k] + W_{x+1,\omega}[k+1, j]\}. \quad (12.10)$$

In this case, all codewords start with the letter a . Moreover, the second letter cannot be the same for all codewords (otherwise this letter can be removed and this improves the solution). Finally, for $a \leq b$, the boundary conditions are:

$$W_{a,b}[i, i] = \min_{a \leq x \leq b} \{W_{x,x}[i, i]\}, \quad W_{x,x}[i, i] = \text{cost}(x) \text{weight}(i). \quad (12.11)$$

The appropriate way to compute the W 's is by increasing values of the difference $j - i$, starting with (12.11) and, for a fixed value of $j - i$, by increasing lengths of the source alphabet intervals, starting with (12.10), followed by (12.9).

This method gives an algorithm running in time $O(q^2 n^3)$ where q is the size of the channel alphabet. Indeed, each evaluation of (12.10) requires

time $O((j - i)q)$, and each evaluation of (12.9) is done in time $O(j - i)$. Itai [40] has given an improvement of the algorithm that leads to a better bound of $O(q^2n^2)$.

Example 12.12. Consider a five symbol source alphabet and a three letter channel alphabet $\{a, b, c\}$ with weights and costs as follows:

i	1	2	3	4	5
weight(i)	5	8	2	10	4

x	a	b	c
cost(x)	1	3	2

The algorithm computes the following tables:

$$W_{a,a} = \begin{pmatrix} 5 & 34 & 48 & 85 & 115 \\ - & 8 & 22 & 54 & 84 \\ - & - & 2 & 34 & 56 \\ - & - & - & 10 & 32 \\ - & - & - & - & 4 \end{pmatrix}, \quad W_{b,b} = \begin{pmatrix} 15 & 60 & 78 & 135 & 173 \\ - & 24 & 42 & 94 & 132 \\ - & - & 6 & 58 & 88 \\ - & - & - & 30 & 60 \\ - & - & - & - & 12 \end{pmatrix},$$
$$W_{c,c} = \begin{pmatrix} 10 & 47 & 63 & 110 & 144 \\ - & 16 & 32 & 74 & 108 \\ - & - & 4 & 46 & 72 \\ - & - & - & 20 & 46 \\ - & - & - & - & 8 \end{pmatrix}, \quad W_{a,b} = \begin{pmatrix} 5 & 29 & 40 & 78 & 97 \\ - & 8 & 14 & 52 & 66 \\ - & - & 2 & 32 & 46 \\ - & - & - & 10 & 22 \\ - & - & - & - & 4 \end{pmatrix},$$
$$W_{b,c} = \begin{pmatrix} 10 & 31 & 47 & 89 & 123 \\ - & 16 & 28 & 62 & 88 \\ - & - & 4 & 26 & 52 \\ - & - & - & 20 & 38 \\ - & - & - & - & 8 \end{pmatrix}, \quad W_{a,c} = \begin{pmatrix} 5 & 21 & 33 & 60 & 86 \\ - & 8 & 12 & 34 & 60 \\ - & - & 2 & 22 & 40 \\ - & - & - & 10 & 18 \\ - & - & - & - & 4 \end{pmatrix}.$$

Hence, the minimal weight of an encoding is $W_{a,c}[1, 5] = 86$. Since $W_{a,a}[1, 2] + W_{b,c}[3, 5] = W_{a,a}[1, 3] + W_{b,c}[4, 5] = 86$, there are, by (12.9), two optimal trees. Inspection of the matrices yields the trees given in Fig. 12.12.

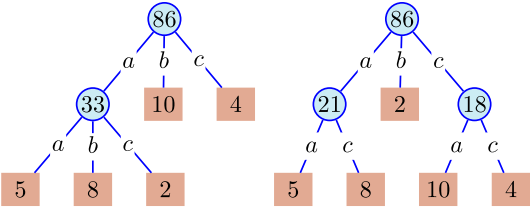


Fig. 12.12. Trees built with Itai’s algorithm.

Optimal encodings with equal weights. In the case where all source symbols have the same weight (vertices 5–8 in Fig. 12.5), this weight can be assumed to be 1. The weighted cost becomes simply

$$W(\gamma) = \sum_{b \in B} \text{cost}(\gamma(b)).$$

The prefix coding problem in this case is known as the *Varn coding problem*. It has an amazingly simple $O(n \log n)$ time solution [67].

We assume that A is a k -letter alphabet and that $n = q(k - 1) + 1$ for some integer q . Hence, the prefix code (or the tree) obtained is complete with q internal nodes and n leaves. Varn's algorithm starts with a tree composed solely of its root, and iteratively replaces a leaf of minimal cost by an internal node, which has k leaves, one for each letter. The number of leaves increases by $k - 1$; hence, in q steps, one gets a tree with n leaves. Note that this solves also the cases numbered 5, 6 in Fig. 12.5.

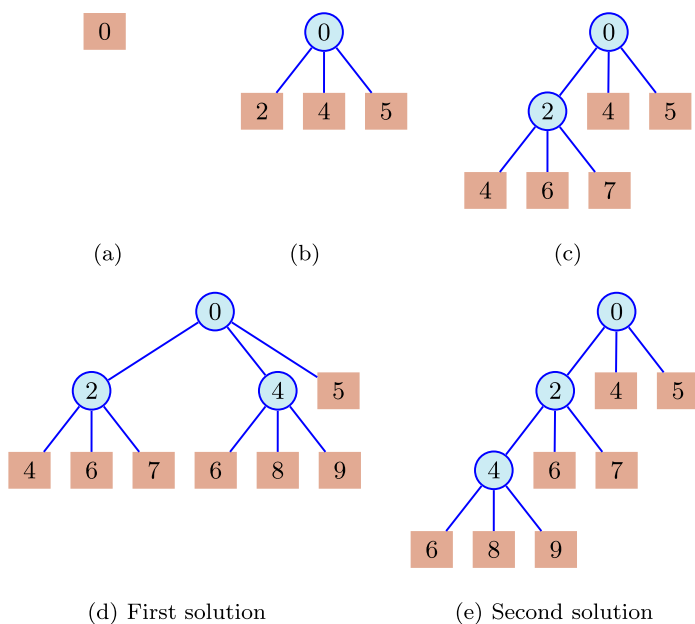


Fig. 12.13. Varn's algorithm for seven words over a 3-letter alphabet. At each step, a leaf of minimal cost is replaced by a node with all possible leaves. There are two choices for the last step. Both give a minimal tree.

Example 12.13. Assume we are looking for a code with seven words over the ternary alphabet $\{a, b, c\}$, and that the cost for letter a is 2, for letter b is 4, and for letter c is 5. The algorithm starts with a tree composed of a single leaf, and then builds the tree by iteration. There are two solutions, both of cost 45, given in Fig. 12.13. Tree 12.13(d) defines the prefix code $\{aa, ab, ac, ba, bb, bc, c\}$, and tree 12.13(e) gives the code $\{aaa, aab, aac, ab, ac, b, c\}$.

12.6.1. Exercises

Exercise 12.6.1. Show that for any distribution p_i of probabilities, the sequence $\ell_i = \lceil \log 1/p_i \rceil$ satisfies the inequality $\sum 2^{-\ell_i} \leq 1$. Conclude that for any source with equal letter costs, there is a prefix code with weighted cost $W \leq H + 1$ where H is the entropy of the source with probabilities p_i .

Exercise 12.6.2. Let A be a k -letter alphabet. A k -ary tree is *complete* if each of its nodes has 0 or k children. A prefix code is complete if its tree is complete. Let C be a finite complete prefix code and let P be the set of prefixes of the words of C , which are not in C . Show that

$$PA \cup \{\varepsilon\} = P \cup C.$$

Deduce that

$$\text{Card}(C) - 1 = \text{Card}(P)(k - 1),$$

where, for a finite set S , $\text{Card}(S)$ denotes the cardinality of S .

Exercise 12.6.3. For a non-empty binary word s of length p , denote by Q the set of words w of length strictly less than p such that sw has s as a suffix. Let X be the set of binary words which have s as a suffix but no other factor is equal to s . Show that X is a maximal prefix code and that the generating series of the lengths of X is

$$f_X(z) = \frac{z^p}{z^p + (1 - 2z)f_Q(z)}, \quad (12.12)$$

where $f_Q(z)$ is the generating series of the lengths of Q (called the *autocorrelation polynomial* of s).

Exercise 12.6.4. Show that, for $s = 101$, one has

$$f_X(z) = \frac{z^3}{1 - 2z + z^2 - z^3}.$$

Table 12.2. The cardinalities of the codes $C_{n,s}$.

	11	10	111	110	101	1111	1110	1010	1001
3	1	2	0	1	1				
4	1	3	1	2	1	0	1	0	1
5	2	4	1	4	2	1	2	2	2
6	3	5	2	7	4	1	4	3	3
7	5	6	4	12	7	2	8	4	6
8	8	7	7	20	12	4	15	9	11
9	13	8	13	33	21	8	28	18	21
10	21	9	24	54	37	15	52	32	39
11	34	10	44	88	65	29	96	60	73
12	55	11	81	143	114	56	177	115	136

Exercise 12.6.5. A set of binary words is said to be a *prefix synchronized code* if all words have the same length n and share a common prefix s which does not appear elsewhere in the sequences of codewords. For each s and n , there is a maximal set $C_{n,s}$, which satisfies this condition. For example, $C_{5,101} = \{10100, 10111\}$. Table 12.2 shows the cardinalities of some of the sets $C_{n,s}$.

Let U be the set of binary words u such that s is a proper prefix of u and us does not have s as a factor except as a prefix or a suffix. Show that

$$f_U(z) = \frac{2z - 1}{z^p + (1 - 2z)f_Q(z)} + 1,$$

where p is the length of s and $f_Q(z)$ is the autocorrelation polynomial of the word s . (Note: Use the fact that $s \cup \{0, 1\}X = X \cup Us$ where X is as in Exercise 12.6.3.) Show that for each $n \geq p$, $C_{n,s}$ is the set of words of length n in U .

Exercise 12.6.6. Let π be a Bernoulli distribution on the source alphabet B . A *Tunstall code* of order n is a maximal prefix code with n codewords over the alphabet B , which has maximal average length with respect to π . Such a code is used to encode the words of the source alphabet by binary blocks of length k for $n \leq 2^k$. For example, if $B = \{a, b\}$ and $\pi(a) = 0.8$, $\pi(b) = 0.2$, the code $C = \{aaa, aab, ab, b\}$ is a Tunstall code of order 4. Its average length is 2.44 and thus coding each word of C by a binary block of length 2 realizes a compression with rate $2/2.44 \approx 0.82$.

Show how Varn's algorithm can be used to build a Tunstall code. (Tunstall codes were introduced in [66], see also [58].)

12.7. Prefix Codes for Integers

Some particular codes are used for compression purposes to encode numerical data subject to a known probability distribution. They appear in particular in the context of digital audio and video coding. The data encoded are integers and thus these codes are infinite. We will consider several families of these codes, beginning with the Golomb codes introduced in [30]. We have already seen the Elias code, which belongs to one of these families.

Golomb codes. The *Golomb code* of order $m \geq 1$, denoted by G_m , is the maximal infinite prefix code:

$$G_m = 1^*0R_m.$$

Thus, words in G_m are composed of a (possibly empty) block of 1's, followed by 0, and followed by a word in R_m , where the prefix codes R_m are defined as follows. If $m = 2^k$ is a power of 2, then R_m is the set of all binary words of length k . In particular, R_1 is composed of the empty word only. For other values of m , the description is more involved. Set $m = 2^k + \ell$, with $0 < \ell < 2^k$. Setting $n = 2^{k-1}$,

$$R_m = \begin{cases} 0R_\ell \cup 1R_{2n} & \text{if } \ell \geq n, \\ 0R_n \cup 1R_{n+\ell} & \text{otherwise.} \end{cases}$$

The codes R_m for $m = 1$ to 7 are represented in Fig. 12.14. Note that, in particular, the lengths of the codewords differ at most by one.

The Golomb codes of order 1, 2, 3 are represented in Fig. 12.15. The encoding of the integers is alphabetic. Note that, except possibly for the first level, there are exactly m words of each length. One way to define directly the encoding of an integer is as follows. Set $r = \lceil \log m \rceil$. Define the *adjusted* binary representation of an integer $n < m$ as its representation on $r - 1$ bits if $n < 2^r - m$ and on r bits otherwise (adding 0's on the left if necessary).

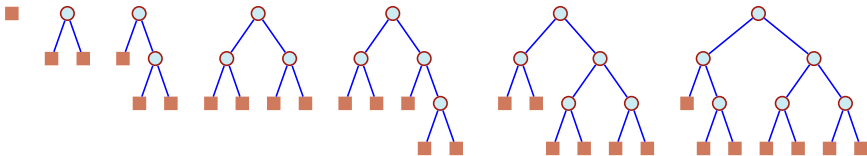


Fig. 12.14. The sets R_1 to R_7 .

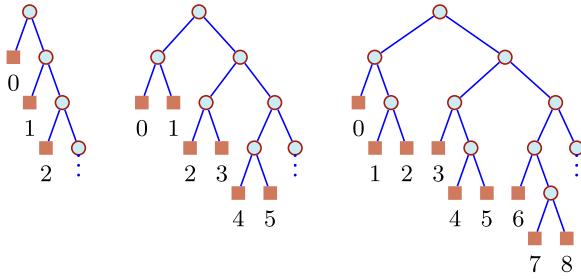


Fig. 12.15. The Golomb codes of orders 1, 2, 3.

The encoding of the integer n in G_m is formed of n/m 1's followed by 0, followed by the adjusted binary representation of n modulo m .

A *geometric distribution* on the set of integers is given by

$$\pi(n) = p^n q, \quad (12.13)$$

for positive real numbers p, q with $p + q = 1$. Such a distribution may arise from *run-length encoding* where a sequence $0^n 1$ is encoded by n . If the source produces 0 and 1's independently with probability p and q , the probability of $0^n 1$ is precisely $\pi(n)$. This is of practical interest if p is large since then long runs of 0 are expected and the run-length encoding realizes a logarithmic compression.

We will show that for a source of integers with the geometric distribution corresponding to a given p , there is an integer $m \geq 1$ such that the Golomb code G_m is an optimal prefix code.

For this, consider, following [22], the integer m such that

$$p^m + p^{m+1} \leq 1 < p^m + p^{m-1}. \quad (12.14)$$

For each p with $0 < p < 1$, there is a unique integer m satisfying (12.14). Indeed, (12.14) is equivalent to

$$p^m(1+p) \leq 1 < p^{m-1}(1+p)$$

or equivalently

$$m \geq -\frac{\log(1+p)}{\log p} > m-1.$$

Note that when m is large, (12.14) implies $p^m \sim 1/2$, and that (12.14) holds for $p^m = 1/2$. Let us show that the application of the Huffman algorithm to a geometric distribution given by (12.13) can produce the Golomb code

of order m where m is defined by (12.14). This shows the optimality of the Golomb code. Actually, we will operate on a truncated, but growing source since Huffman's algorithm works only on finite alphabets.

Set $Q = 1 - p^m$. By the choice of m , one has $p^{-1-m} \geq 1/Q > p^{1-m}$. We consider, for $k \geq -1$, the bounded alphabet

$$B_k = \{0, \dots, k + m\}.$$

In particular, $B_{-1} = \{0, \dots, m - 1\}$. We consider on B_k the distribution

$$\pi(i) = \begin{cases} p^i q & \text{for } 0 \leq i \leq k, \\ p^i q / Q & \text{for } k < i \leq k + m. \end{cases}$$

Clearly $\pi(i) > \pi(k)$ for $i < k$ and $\pi(k + i) > \pi(k + m)$ for $1 < i < m$. Observe that also $\pi(i) > \pi(k + m)$ for $i < k$ since $\pi(k + m) = p^{k+m}q/Q \leq p^{k+m}q/p^{m+1} = \pi(k - 1)$. Also $\pi(k + i) > \pi(k)$ for $1 < i < m$ since indeed $\pi(k + i) > \pi(k + m - 1) = p^{k+m-1}q/Q > p^k q = \pi(k)$. As a consequence, the symbols k and $k + m$ are those of minimal weight. Huffman's algorithm replaces them with a new symbol, say k' , which is the root node of a tree with, say, left child k and right child $k + m$. The weight of k' is

$$\pi(k') = \pi(k) + \pi(k + m) = p^k q (1 + p^m / Q) = p^k q / Q.$$

Thus, we may identify $B_k \setminus \{k, k + m\} \cup \{k'\}$ with B_{k-1} by assigning to k the new value $\pi(k) = p^k q / Q$. We get for B_{k-1} the same properties as for B_k and we may iterate.

After m iterations, we have replaced B_k by B_{k-m} , and each of the symbols $k - m + 1, \dots, k$ now is the root of a tree with two children. Assume now that $k = (h + 1)m - 1$ for some h . Then after hm steps, one gets the alphabet $B_{-1} = \{0, \dots, m - 1\}$, and each of the symbols i in B_{-1} is the root of a binary tree of height h composed of a unique right path of length h , and at each level one left child $i + m, i + 2m, \dots, i + (h - 1)m$. This corresponds to the code $P_h = \{0, 10, \dots, 1^{h-1}0, 1^h\}$. The weights of the symbols in B_{-1} are decreasing, and moreover $\pi(m - 2) + \pi(m - 1) > \pi(0)$ because $p^{m-2} + p^{m-1} > 1$. It follows from Exercise 12.7.2 below that the code R_m is optimal for this probability distribution.

Thus we have shown that the application of Huffman's algorithm to the truncated source produces the code $R_m P_k$. When h tends to infinity, the sequence of codes converges to $R_m 1^*0$. Since each of the codes in the sequence is optimal, the code $R_m 1^*0$ is an optimal prefix code for the

geometric distribution. The Golomb code $G_m = 1^*0R_m$ has the same length distribution and so is also optimal.

Golomb–Rice codes. The *Golomb–Rice code* of order k , denoted GR_k , is the particular case of the Golomb code for $m = 2^k$. It was introduced in [54]. Its structure is especially simple and allows an easy explicit description. The encoding assigns to an integer $n \geq 0$ the concatenation of two binary words, the *base* and the *offset*. The base is the unary expansion (over the alphabet $\{1\}$) of $\lfloor n/2^k \rfloor$ followed by a 0. The offset is the remainder of the division written in binary on k bits. Thus, for $k = 2$, the integer $n = 9$ is coded by 110|01. The binary trees representing the Golomb–Rice code of orders 0, 1, 2 are represented in Fig. 12.16.

Another description of the Golomb–Rice code of order k is given by the regular expression

$$GR_k = 1^*0\{0,1\}^k. \quad (12.15)$$

This indicates that the binary words forming the code are composed of a base of the form 1^i0 for some $i \geq 0$ and an offset, which is an arbitrary binary sequence of length k .

It follows from (12.15) that the generating series of the lengths of words of the Golomb–Rice code of order k is

$$f_{GR_k}(z) = \frac{2^k z^{k+1}}{1-z} = \sum_{i \geq k+1} 2^k z^i.$$

The *weighted generating series* of a code C with probability distribution π is

$$p_C(z) = \sum_{x \in C} \pi(x) z^{|x|}.$$

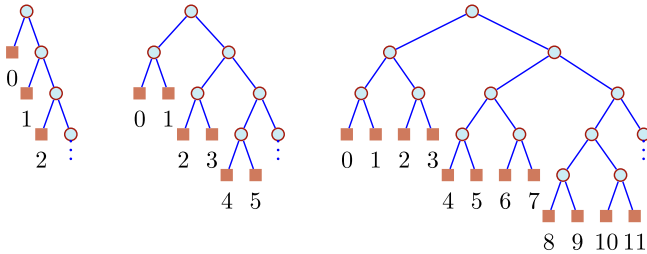


Fig. 12.16. The Golomb–Rice codes of orders 0, 1, 2.

The average length of C is then

$$\lambda_C(z) = \sum_{x \in C} |x| \pi(x) = p'_C(1).$$

For a uniform Bernoulli distribution on the channel symbols, the weighted generating series for the resulting probabilities of the Golomb–Rice codes GR_k and the corresponding average length λ_{GR_k} are

$$\begin{aligned} p_{GR_k}(z) &= f_{GR_k}\left(\frac{z}{2}\right) = \frac{z^{k+1}}{2-z}, \\ \lambda_{GR_k} &= p'_{GR_k}(1) = k+2. \end{aligned} \tag{12.16}$$

Observe that in the case where $p^m = 1/2$, the series $p_{GR_k}(z)$, and thus also the average length λ_{GR_k} , happens to be the same for the probability distribution on the code induced by the geometric distribution on the source.

Indeed, the sum of the probabilities of the codewords for rm to $(r+1)m-1$ is $p^{rm}(1-p^m)$, and since $p^m = 1/2$, this is equal to 2^{-r-1} . The sum of the probabilities of m words of length $r+k+1$ with respect to a uniform Bernoulli distribution is $m2^{-r-k-1} = 2^{-r-1}$ (recall that $m = 2^k$). Thus, we get the same value in both cases, as claimed.

Exponential Golomb codes. The *exponential Golomb codes*, denoted EG_k for $k \geq 0$, form a family of codes whose length distributions make them better suited than the Golomb–Rice codes for encoding the integers endowed with certain probability distributions. They are introduced in [64]. The case $k = 0$ is the *Elias code* already mentioned and introduced in [19]. Exponential Golomb codes are used in practice in digital transmissions. In particular, they are a part of the video compression standard technically known as H.264/MPEG-4 Advanced Video Coding (AVC) [55].

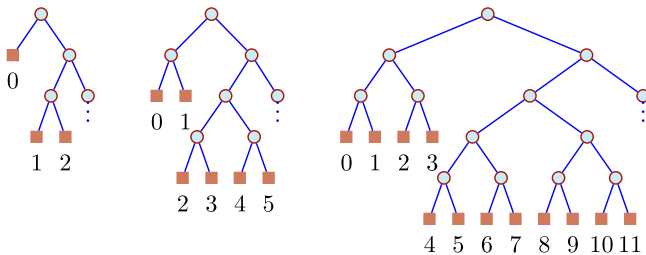


Fig. 12.17. The exponential Golomb codes of orders 0, 1, 2.

The codeword representing an integer n tends to be shorter for large integers. The base of the codeword for an integer n is obtained as follows. Let x be the binary expansion of $1 + \lfloor n/2^k \rfloor$ and let i be its length. The base is made of the unary expansion of $i - 1$ followed by x with its initial 1 replaced by a 0. The offset is, as before, the binary expansion of the remainder of the division of n by 2^k , written on k bits. Thus, for $k = 1$, the codeword for 9 is 11001|1. Figure 12.17 represents the binary trees of the exponential Golomb codes of orders 0, 1, 2. An expression describing the exponential Golomb code of order k is

$$EG_k = \bigcup_{i \geq 0} 1^i 0 \{0, 1\}^{i+k},$$

and we have the simple relation

$$EG_k = EG_0 \{0, 1\}^k.$$

The generating series of the lengths of words in EG_k is

$$f_{EG_k}(z) = \frac{2^k z^{k+1}}{1 - 2z^2}.$$

The weighted generating series for the probabilities of codewords corresponding to a uniform Bernoulli distribution and the average length are

$$p_{EG_k}(z) = \frac{z^{k+1}}{2 - z^2},$$

$$\lambda_{EG_k} = k + 3.$$

For handling signed integers, there is a simple method, which consists of adding a bit to the words of one of the previous codes. More sophisticated methods, adapted to a two-sided geometric distribution, have been developed in [50].

12.7.1. Exercises

Exercise 12.7.1. Show that the entropy $H = -\sum \pi(n) \log \pi(n)$ of the source emitting integers with a geometric distribution (12.13) is

$$H = -(p \log p + q \log q)/q.$$

Verify directly that, for $p^{2^k} = 1/2$, the average length of the Golomb–Rice code GR_k satisfies $\lambda_{GR_k} \leq H + 1$.

Exercise 12.7.2. Let $m \geq 3$. A non-increasing sequence $w_1 \geq w_2 \geq \dots \geq w_m$ of integers is said to be *quasi-uniform* if it satisfies $w_{m-1} + w_m \geq w_1$. Let T be an optimal binary tree corresponding to a quasi-uniform sequence of weights. Show that the heights of the leaves of T differ at most by one.

12.8. Encoders and Decoders

In this section, we present the basic notions of automata theory, as far as encoding and decoding processes are concerned. The notion of finite automata allows us to define a state-dependent process, which can be used for both encoding and decoding. We give the definition of two closely related notions, namely automata and transducers, which are both labeled directed graphs.

A *finite automaton* \mathcal{A} over some (finite) alphabet A is composed of a finite set Q of *states*, together with two distinguished subsets I and T called the sets of *initial* and *terminal* states, and a set E of *edges*, which are triples (p, a, q) where p and q are states and a , is a symbol. An edge is also denoted by $p \xrightarrow{a} q$. It starts in p , ends in q , and has label a .

Similarly, a *finite transducer* \mathcal{T} uses an input alphabet A and an output alphabet B . It is composed of a finite set Q of *states*, together with two distinguished subsets I and T called the sets of *initial* and *terminal* states, and a set E of *edges*, which are quadruples (p, u, v, q) where p and q are states and u is a word over A and v is a word over B . An edge is also denoted by $p \xrightarrow{u|v} q$. The main purpose for transducers is decoding. In this case, A is the channel alphabet and B is the source alphabet.

A path in an automaton or in a transducer is a finite sequence of consecutive edges. The label of the path is obtained by concatenating the labels of the edges (in the case of a transducer, one concatenates separately the input and the output labels). We write $p \xrightarrow{w} q$ for a path in an automaton labeled with w starting in state p and ending in state q . Similarly, we write $p \xrightarrow{x|y} q$ for a path in a transducer. A path is *successful* if it starts in an initial state and ends in a terminal state.

An automaton \mathcal{A} *recognizes* a set of words, which is the set of labels of its successful paths. The sets recognized by finite automata are called *regular sets*.

A transducer \mathcal{T} defines a binary relation between words on the two alphabets as follows. A pair (u, v) is in the relation if it is the label of a successful path. This is called the relation *realized* by \mathcal{T} . This relation can be viewed as a multi-valued mapping from the input words into the output

words, and also as a multi-valued mapping from the output words into the input words. For practical purposes, this definition is too general and will be specialized. We consider transducers called *literal*, which by definition means that each input label is a single letter. For example, an encoding γ , as defined at the beginning of the chapter, can be realized by a one-state literal transducer, with the set of labels of edges being simply the pairs $(b, \gamma(b))$ for b in B .

Example 12.14. Consider the encoding defined by $\gamma(a) = 00$, $\gamma(b) = 1$, and $\gamma(c) = 01$. The corresponding encoding transducer is given in Fig. 12.18.

Transducers for decoding are more interesting. For the purpose of coding and decoding, we are concerned with transducers which define single-valued mappings in both directions. We need two additional notions.

An automaton is called *deterministic* if it has a unique initial state and if, for each state p and each letter a , there is at most one edge starting in p and labeled with a . This implies that, for each state p and each word w , there exists at most one path starting in p and labeled with w .

Consider a finite deterministic automaton with a unique terminal state, which is equal to the initial state i . The closed paths from i to i such that no initial segment ends in i are called *first return paths*. The set of labels of these paths is a regular prefix code C (that is a prefix code, which is a regular set), and the set recognized by the automaton is the set C^* . Conversely, any regular prefix code is obtained in this way.

For example, the Golomb codes are regular, whereas exponential Golomb codes are not.

More generally, an automaton is called *unambiguous* if, for all states p, q , and all words w , there is at most one path from p to q labeled with w . Clearly, a deterministic automaton is unambiguous.

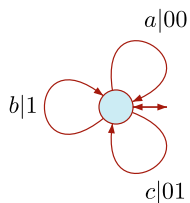


Fig. 12.18. A simple encoder. The only state is both initial and terminal.

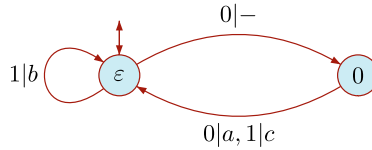


Fig. 12.19. A deterministic decoder. A dash represents the empty word. An incoming (outgoing) arrow indicates the initial (terminal) state.

A literal transducer defines naturally an automaton over its input alphabet, called its *input automaton*. For simplicity, we discard the possibility of multiple edges in the resulting automaton. A literal transducer is called *deterministic* (resp. *unambiguous*) if its associated input automaton is deterministic (resp. unambiguous). Clearly, the relation realized by a deterministic transducer is a function.

An important result is that for any encoding (with finite source and channel alphabets), there exists a literal unambiguous transducer, which realizes the associated decoding. When the code is prefix, the transducer is actually deterministic.

The construction is as follows. Let γ be an encoding. Define a transducer \mathcal{T} by taking a state for each proper prefix of some codeword. The state corresponding to the empty word ε is the initial and terminal state. There is an edge $p \xrightarrow{a|-} pa$, where $-$ represents the empty word, for each prefix p and letter a such that pa is a prefix, and an edge $p \xrightarrow{a|b} \varepsilon$ for each p and letter a with $pa = \gamma(b)$. When the code is prefix, the decoder is deterministic. In the general case, the property of unique decipherability is reflected by the fact that the transducer is unambiguous.

Example 12.15. The decoder corresponding to the prefix code of Example 12.14 is represented in Fig. 12.19.

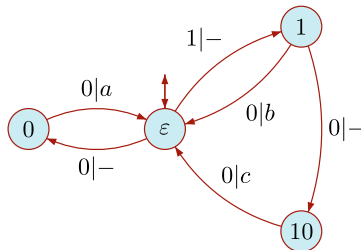


Fig. 12.20. An unambiguous decoder for a code, which is not prefix.

Example 12.16. Consider the code $C = \{00, 10, 100\}$ of Example 12.4. The decoder given by the construction is represented in Fig. 12.20.

As a consequence of this construction, it can be shown that decoding can always be realized in linear time with respect to the length of the encoded string (considering the number of states as a constant). Indeed, given a word $w = a_1 \cdots a_n$ of length n to be decoded, one computes the sequence of sets S_i of states accessible from the initial state for each prefix $a_1 \cdots a_i$ of length i of w , with the convention $S_0 = \{\varepsilon\}$. Of course, the terminal state ε is in S_n . Working backwards, we set $q_n = \varepsilon$ and we identify in each set S_i the unique state q_i such that there is an edge $q_i \xrightarrow{a_i} q_{i+1}$ in the input automaton. The uniqueness comes from the unambiguity of the transducer. The corresponding sequence of output labels gives the decoding. This construction is based on the *Schützenberger covering* of an unambiguous automaton, see [57].

Example 12.17. Consider again the code $C = \{00, 10, 100\}$. The decoding of the sequence 10001010000 is represented on Fig. 12.21. Working from left to right produces the tree of possible paths in the decoder of Fig. 12.20. Working backwards from the state ε in the last column produces the successful path indicated in boldface.

The notion of deterministic transducer is too constrained for the purpose of coding and decoding because it does not allow a lookahead on the input or equivalently a delay on the output. The notion of sequential transducer to be introduced now fills this gap.

A *sequential transducer* is composed of a deterministic transducer and an output function. This function maps the terminal states of the transducer into words on the output alphabet. The function realized by a sequential transducer is obtained by appending, to the value of the deterministic transducer, the image of the output function on the arrival state. Formally, the value on the input word x is

$$f(x) = g(x)\sigma(i \cdot x),$$

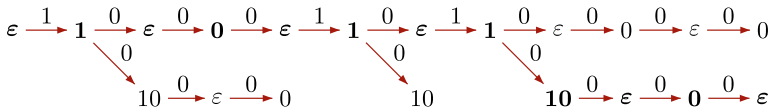


Fig. 12.21. The decoding of 10001010000.

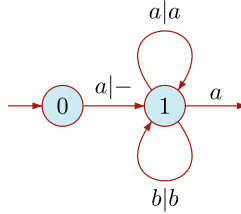


Fig. 12.22. A sequential transducer realizing a cyclic shift on words starting with the letter a , with $\sigma(0) = \varepsilon$ and $\sigma(1) = a$.

where $g(x)$ is the value of the deterministic transducer on the input word x , where $i \cdot x$ is the state reached from the input state i by the word x , and where σ is the output function. This is defined only if the state $i \cdot x$ is a terminal state.

Example 12.18. The sequential transducer given in Fig. 12.22 realizes the partial function $aw \mapsto wa$, for each word w . The output function σ is given by $\sigma(0) = \varepsilon$ and $\sigma(1) = a$.

It is a well-known property of finite automata that any finite automaton is equivalent to a finite deterministic automaton. The process realizing this transformation is known as the *determinization algorithm*. This remarkable property does not hold in general for transducers.

Nonetheless, there is an effective procedure to compute a sequential transducer \mathcal{S} that is equivalent to a given literal transducer \mathcal{T} , whenever such a transducer exists, see [46]. The algorithm goes as follows.

The states of \mathcal{S} are sets of pairs (u, p) . Each pair (u, p) is composed of an output word u and a state p of \mathcal{T} . For a state s of \mathcal{S} and an input letter a , one first computes the set \bar{s} of pairs (uv, q) such that there is a pair (u, p) in s and an edge $p \xrightarrow{a|v} q$ in \mathcal{T} . In a second step, one chooses some common prefix z of all words uv , and one defines the set $t = \{(w, q) \mid (zw, q) \in \bar{s}\}$. There is a transition from state s to a state t labeled with (a, z) . The initial state is (ε, i) , where i is the initial state of \mathcal{T} . The terminal states are the sets t containing a pair (u, q) with q terminal in \mathcal{T} . The output function σ is defined on state t of \mathcal{S} by $\sigma(t) = u$. If there are several pairs (u, q) in t with distinct u for the same terminal state q , then the given transducer does not compute a function and thus it is not equivalent to a sequential one.

The process of building new states of \mathcal{S} may not halt if the lengths of the words, which are the components of the pairs, are not bounded.

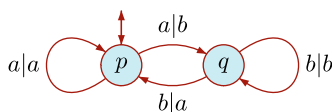


Fig. 12.23. Another transducer realizing a cyclic shift on words starting with the letter a .

There exist *a priori* bounds for the maximal length of the words appearing whenever the determinization is possible, provided that, at each step, the longest common prefix is chosen. This makes the procedure effective.

Example 12.19. Consider the transducer given in Fig. 12.23. The result of the determinization algorithm is the transducer of Fig. 12.22. State 0 is composed of the pair (ε, p) , and state 1 is formed of the pairs (a, p) and (b, q) .

Example 12.20. Consider the code $C = \{00, 10, 100\}$ of Example 12.4. Its decoder is represented in Fig. 12.20. The determinization algorithm applied to this transducer produces, for the input word 10^{2n} , the state consisting of $(ba^{n-1}, 0)$ and (ca^{n-1}, ε) . Thus, the algorithm does not terminate, and there is no equivalent sequential transducer.

12.8.1. Exercises

Exercise 12.8.1. A set of words C is said to be *weakly prefix* if there is an integer $d \geq 0$ such that the following condition holds for any elements c, c' of C and any words w, w' of C^* . If the prefix of length $|c| + d$ of cw is a prefix of $c'w'$, then $c = c'$. The least integer d such that this property holds is called the *deciphering delay* and a weakly prefix code is also said to have *finite deciphering delay*.

Show that a weakly prefix set is UD.

Exercise 12.8.2. Which of the following sets C and C' is a weakly prefix code? $C = \{00, 10, 100\}$ and $C' = \{0, 001\}$.

Exercise 12.8.3. Show that a finite code is weakly prefix if and only if it can be decoded by a sequential transducer.

12.9. Codes for Constrained Channels

The problem considered in this section arises in connection with the use of communication channels which impose input constraints on the

sequences that can be transmitted. User messages are encoded to satisfy the constraint; the encoded messages are transmitted across the channel and then decoded by an inverse to the encoder.

In this context, we will use a more general notion of encoding. Instead of a memoryless substitution of source symbols by codewords, we will consider finite transducers: the codeword associated with a source symbol depends not only on this symbol but also on a state depending on the past.

We require, in order to be able to decode, that the encoder is unambiguous on its output in the sense that for each pair of states and each word w , there is at most one path between these states with output label w . For ease of use, we will also assume that the input automaton of the encoder is deterministic.

Example 12.21. The encoder in Fig. 12.26 is deterministic. The source alphabet is $\{0, 1\}$ and the channel alphabet is $\{a, b, c\}$. The sequence 0011 is encoded by $acbb$ if the encoding starts in state 1. A more complicated example of a deterministic encoder that will be described later is depicted in Fig. 12.31.

A stronger condition for a decoder than unambiguity is that of having *finite look-ahead*. This means that there exists an integer $D \geq 0$ such that, for all states q and all channel sequences w of length $D + 1$, all paths that begin at state q and have channel label w share the same first edge (and therefore the first source symbol). In other words, decoding is a function of the current state, the current channel symbol, and the upcoming string of D channel symbols [45]. These decoders correspond to sequential transducers as introduced in Sec. 12.8.

However, even this condition is heavily state-dependent, and so a channel error that causes the decoder to lose track of the current state may propagate errors forever. For this reason, the following stronger condition is usually employed.

A *sliding block decoder* operates on words of channel symbols with a window of fixed size. The decoder uses m symbols before the current one and a symbols after it (m is for memory and a for anticipation). According to the value of the symbols between time $n - m$ and time $n + a$, the value of the n th source symbol is determined. Figure 12.24 depicts a schematic view of a sliding block decoder. It is not hard to show that sliding-block decodability implies the weaker finite lookahead property mentioned above. Note that a sliding block decoder avoids possible problems with error propagation

because any channel error can affect the decoding only while it is in the decoder window and thus can corrupt at most $m + a + 1$ source symbols.

Transducers realizing sliding block decoders are of a special kind. An automaton is called *local* if the knowledge of a finite number of symbols in the past and in the future determines the current state. More precisely, for given integers $m, a \geq 0$ (m stands for memory and a for anticipation), an automaton is said to be (m, a) -local if for words u and v of length m and a , respectively, $p \xrightarrow{u} q \xrightarrow{v} r$ and $p' \xrightarrow{u} q' \xrightarrow{v} r'$ imply that $q = q'$. A transducer is said to be *local* if its input automaton is local.

A sliding block decoder can be realized by a local transducer with the same parameters. Conversely, a transducer whose input automaton is (m, a) -local, and such that the input label and the output label of each edge have the same length, is a sliding block decoder.

While input constraints have been imposed in some communication channels, they arise more commonly in data recording channels, such as those found in magnetic and optical disk drives. Constraints on inputs are of various types and have changed over the course of the 50-year history of magnetic disk drives. For illustration, we focus on the $[d, k]$ -constraint, where $0 \leq d \leq k$. A binary sequence is said to satisfy the $[d, k]$ -constraint if the number of contiguous symbols 0 between consecutive symbols 1 is at least d and at most k .

These constraints arise for the following reasons. An electrical current in the write head, situated over the spinning disk, creates a magnetic field, which is reversed when the current polarity is reversed. These write field reversals, in turn, induce reversals in the orientation of the magnetization along the recorded track on the disk. During the data recovery process, the read head senses the recorded pattern of magnetization along the track. Each transition in magnetization direction produces a correspondingly oriented pulse in the readback voltage.

Two problems should be avoided. The first is called *intersymbol interference*. If polarity changes are too close together, the induced magnetic

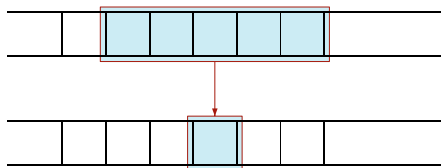


Fig. 12.24. A sliding block decoder.

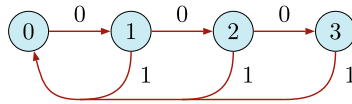


Fig. 12.25. The $[1, 3]$ -constraint.

fields tend to interfere and the pulses in the readback signal are harder to detect. The second problem is called *clock drift*. This problem arises when the pulses are separated by intervals of time which are too large. When the read head senses a pulse it sends information through a phase lock loop which keeps the bit clock running accurately; if the separation between pulses is too large, the clock can lose synchronization and skip through a complete bit period.

Several values of the parameters d, k are of practical importance. The simplest case is the constraint $[1, 3]$. This means that the blocks 11 and 0000 are forbidden. The binary sequences satisfying this constraint are those which label paths in the graph of Fig. 12.25.

We consider an encoding of all binary sequences by sequences satisfying the $[1, 3]$ -constraint. This encoding is not realizable by a sequential encoder without modifying the output alphabet, because there are more binary source sequences of length n than admissible binary channel sequences of length n . However, it is possible to operate at rate 1:2, by encoding a source bit by one of the 2-bit symbols $a = 00$, $b = 01$, or $c = 10$. A particular way of doing this is the modified frequency modulation (MFM, see [47]) code of Fig. 12.26, which was used on floppy disks for many years.

Observe that the second bit of the output is always equal to the input bit, and so the decoder can operate symbol by symbol, producing a 1-bit input from a 2-bit output. The first bit of the output is chosen in such a way that there are no consecutive 1's and no block of four 0's.

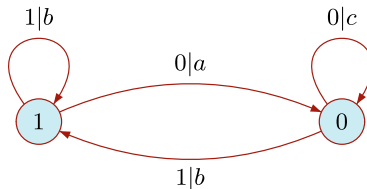
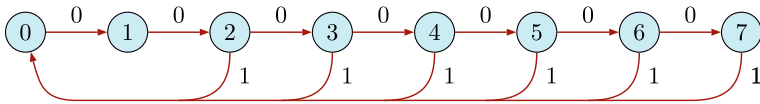
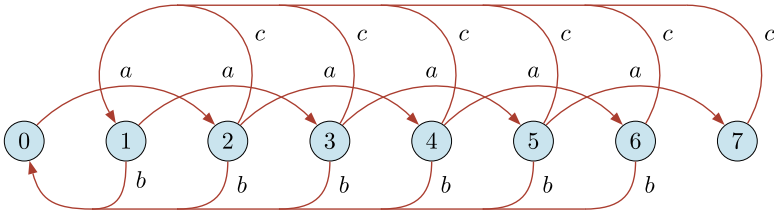


Fig. 12.26. The MFM code. Here the state names reflect the input bit.

Fig. 12.27. The $[2, 7]$ -constraint.Fig. 12.28. The squared $[2, 7]$ -constraint.

A more complex example is the $[2, 7]$ -constraint illustrated in Fig. 12.27. Again, the sequences satisfying the constraint are the labels of paths in the graph.

For the purpose of coding arbitrary binary sequences by sequences satisfying the $[2, 7]$ -constraint, we again consider a representation obtained by changing the alphabet to $a = 00$, $b = 01$, and $c = 10$, as shown in Fig. 12.28.

The result of the development that follows will be the sequential transducer for encoding known as the Franaszek encoder depicted in Fig. 12.31. It must be checked that this encoder satisfies the $[2, 7]$ -constraint and that, for practical applications, it admits a sliding block decoder. A direct verification is possible but complicated.

The encoder design process we now describe is the one historically followed by Franaszek [21]. It starts with the graph in Fig. 12.28, which represents the $[2, 7]$ -constraint of Fig. 12.27 in terms of the alphabet a, b, c . More precisely, Fig. 12.28 represents the set of $[2, 7]$ -constrained sequences

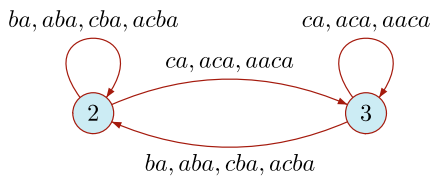


Fig. 12.29. The poles.

C	P
ba	10
ca	11
aba	000
cba	010
aca	011
$acba$	0010
$aaca$	0011

Fig. 12.30. The Franaszek code.

obtained by writing each such sequence as a string of non-overlapping 2-bit pairs.

Next, choose the two vertices 2 and 3 (called the *poles* or the *principal states*). The paths of first return from 2 or 3 to 2 or 3 are represented in Fig. 12.29.

Observe that the set C of labels of first return paths is independent of the starting vertex. Thus all concatenations of words in C are produced by paths in the graph of Fig. 12.29 and thus admissible for the $[2,7]$ -constraint. The set C is a prefix code called the *Franaszek code*, shown in the first column of Fig. 12.30.

It happens that the set C has the same length distribution as the maximal binary prefix code P of words appearing in the right column.

The pair of prefix codes of Fig. 12.30 is used as follows to encode a binary word at rate 1:2. A source message is parsed as a sequence of codewords in P , possibly followed by a prefix of such a word. For example, the word

$$011011100101110111100010011 \dots$$

is parsed as

$$011 \mid 011 \mid 10 \mid 010 \mid 11 \mid 10 \mid 11 \mid 11 \mid 000 \mid 10 \mid 011 \dots$$

Next, this is encoded row-by-row using the correspondence between P and C as follows:

$$aca \mid aca \mid ba \mid cba \mid ca \mid ba \mid ca \mid ca \mid aba \mid ba \mid aca.$$

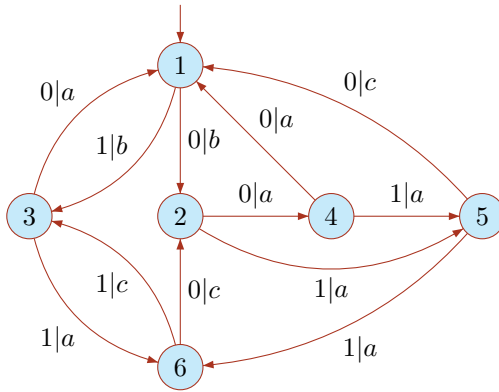


Fig. 12.31. The Franaszek encoder.

This stands for the channel encoding

$$\begin{aligned} &001000 \mid 001000 \mid 0100 \mid 100100 \mid 1000 \mid 0100 \mid 1000 \mid \\ &1000 \mid 000100 \mid 0100 \mid 001000. \end{aligned}$$

Figure 12.31 represents an implementation of this transformation, up to some shift. The encoder is a deterministic transducer which outputs one symbol for each input symbol. State 1 is the initial state, and all states are terminal states. All inputs of at least 2 symbols produce an output sequence that begins with ba , followed by the sequence generated by the encoder described above, up to the last two symbols. For example, for the input word 011011, the corresponding path in the deterministic encoder is

$$1 \xrightarrow{0|b} 2 \xrightarrow{1|a} 5 \xrightarrow{1|a} 6 \xrightarrow{0|c} 2 \xrightarrow{1|a} 5 \xrightarrow{1|a} 6.$$

Hence, the output is $ba|aca|a$, which, following the initial ba , corresponds to one codeword aca followed by the beginning of a second occurrence of aca .

This encoder can be obtained as follows from the pair (C, P) of prefix codes. Consider first the transducer of Fig. 12.32. This is obtained by composing a decoder for P with an encoder for C , and merging common prefixes. We omit the details.

We build a deterministic transducer by determinization of the transducer of Fig. 12.32, by the algorithm presented in Sec. 12.8. In this case, we are able to maintain a constant rate of output by keeping always in the pairs of the sequential transducer an output word of length 2 (this

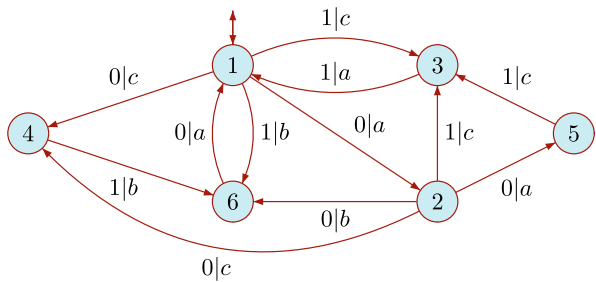


Fig. 12.32. The construction of the Franaszek encoder.

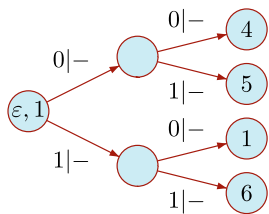


Fig. 12.33. Initial part of the Franaszek encoder.

represents the two symbols to be output later). Thus, the output is delayed with two symbols.

The states of the result are given in Table 12.3. In this table, and also in Fig. 12.31, we only report the states which have an output delay of exactly two symbols. They correspond to the strongly connected part of the encoder. In particular, the initial state $(\varepsilon, 1)$ is not represented. Figure 12.33 gives the states reached by the first two symbols.

Figure 12.34 represents the decoder, which can be realized with a sliding window of size 4. Indeed, the diagrams of Fig. 12.34 show that for any content $xyzt$ of the window, we can define the output r corresponding to the first symbol. An inspection of Figure 12.31 shows that if $y = b$, then $r = 0$. In the same manner, if $y = c$, then $r = 1$. If $y = a$, then four cases arise. If $z = a$, then $r = 0$. If $z = c$, then $r = 0$ or 1 according to $t = b$ or a .

Table 12.3. The states of the Franaszek encoder.

state	1	2	3	4	5	6
content	$ba,1$	$ac,4$ $aa,2$	$ab,6$ $ac,3$	$ac,4$ $ab,6$ $aa,5$	$cb,6$ $ac,3$	$ca,1$
output	ba					ca

Finally, if $z = b$, then $r = 0$ if $z = b$ and $r = 1$ otherwise (the x in the last frame stands for b or c).

There are many approaches to the design of encoder–decoder pairs for input constraints. The approach illustrated above was extended to a technique called the *method of poles*, developed by Béal [6–8].

Another approach is the *state-splitting algorithm*, also known as the *ACH algorithm* [1], which transforms a finite-state graph representation of a constraint into a sliding-block decodable finite-state encoder. The rough idea is as follows.

One starts with a finite-state graph representation of the constraint and then chooses a feasible encoding rate $p:q$; here, feasibility means that p/q does not exceed the capacity of the constraint, which is a measure of the “size” or “complexity” of the constraint and generalizes the notion of capacity of a variable-length code given in Sec. 12.5. The capacity can be computed explicitly, and for a feasible code rate, one can compute a vector, called an approximate eigenvector, which effectively assigns non-negative integer weights to the states of the graph [45]. Next, one replaces the original representation with a new one whose symbols are q -bit strings, just as in the transformation of Figs. 12.27 and 12.28. Then by an iterative sequence of state splittings, guided by an approximate eigenvector, one is guaranteed to arrive at a final representation upon which a sequential finite-state encoder can be built. The splittings are graph transformations in which states are split according to partitions of outgoing edges. In the final representation, each state has at least 2^p outgoing edges, enabling an assignment of p -bit input labels and therefore a sequential finite-state encoder at rate $p:q$. It is also guaranteed, under mild conditions, that such an encoder is sliding-block decodable.

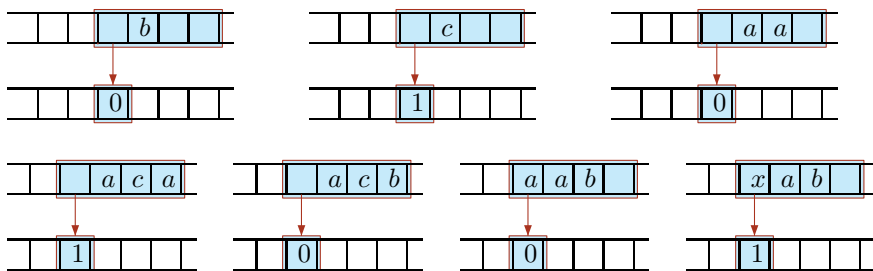


Fig. 12.34. The associated sliding block decoder.

The state splitting algorithm has been modified in many ways. For instance, one could consider finite-state variable-length representations of input constraints or transform fixed-length representations into more compact variable-length representations. There is a variable-length version of the state splitting algorithm, which again iteratively splits states. In this setting, instead of aiming for a graph with sufficient out-degree at each state, the sequence of state splittings results in a final representation in which the lengths of outgoing edges at each state satisfy a reverse Kraft inequality. This technique was introduced in [2] and further developed in [33]. In the latter reference, the method was illustrated with two examples, one of which is an alternative derivation of the Franaszek code described above.

The theory and practice of coding for constrained channels is very well developed. For more information, the reader may consult Béal [7], Immink [39], Lind–Marcus [45], and Marcus–Roth–Siegel [47] (or the latest version at <http://www.math.ubc.ca/~marcus/Handbook/index.html>).

12.9.1. Exercises

Exercise 12.9.1. A code C over the alphabet A is a *circular code* if, for any words u, v on the alphabet A , the cyclically shifted words uv and vu can be in C^* only when u and v are. Let C be a circular code and let x be a word of C . Show that a set D of words of the form $x^i y$, for $i \geq 0$ and y in $C \setminus x$, is a circular code.

Note: The terminology *circular code* stems from the fact that the unique decipherability property holds for words written on a circle (see [10]).

Exercise 12.9.2. Use Exercise 12.9.1 to show that the set $C = \{ba, ca, aba, cba, aca, acba, aaca\}$ appearing in the first column of Fig. 12.30 is a circular code.

12.10. Codes for Constrained Sources

In the same way as there exist codes for constrained channels, there exist codes for constrained sources. As for constraints on channels, the constraints on sources can be expressed by means of finite automata. This leads us to use encodings with memory.

We limit the presentation to two examples, the first drawn from linguistics, and the second reflecting a more general point of view.

The first example gives an interesting encoding from a constrained source to an unconstrained channel [31]. It starts with a simple substitution, which is not UD, but it appears to be UD by taking into account the constraints on the source. The example is taken from the field of natural language processing where codings and automata are heavily used (see, for instance, [56]).

Example 12.22. We start with a source alphabet composed of a sample set of six syllable types in the Turkish language. The concatenation of syllables is subject to constraints that will be explained below. The types of syllables are denoted by A to F as follows (with the syllables written in italics in the example):

Symbol	Structure	Example
A	0	<i>açık</i> (open)
B	10	<i>baba</i> (father)
C	01	<i>ekmek</i> (bread)
D	101	<i>altın</i> (gold)
E	011	<i>erk</i> (power)
F	1011	<i>türk</i> (turkish)

The structure of a syllable is the binary sequence obtained by coding 0 for a vowel and 1 for a consonant. We consider the structure of a syllable as its encoding. The decoding is then to recover the sequence of syllables from the encoding. Thus, the source alphabet is $\{A, \dots, F\}$, the channel alphabet is $\{0, 1\}$, and the encoding is $A \mapsto 0, \dots, F \mapsto 1011$.

There are linguistic constraints on the possible concatenations of syllable types, which come from the fact that a syllable ending with a consonant cannot be followed by one which begins with a vowel. These constraints are summarized by the following matrix, where a 0 in entry x, y means that x cannot be followed by y :

$$M = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{matrix}.$$

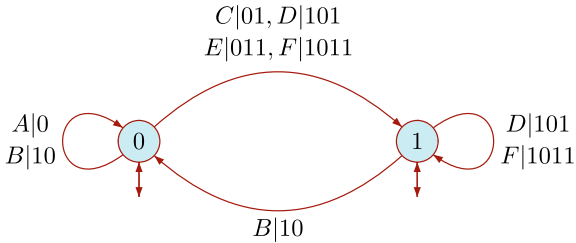


Fig. 12.35. A 2-state encoder reflecting the legal concatenations.

The encoding can be realized in a straightforward manner with a 2-state transducer given in Fig. 12.35, which reflects the legal concatenations of source symbols.

The decoder is built by applying the methods of Sec. 12.8 to this transducer. We start by exchanging input and output labels in Fig. 12.35. In a second step, we introduce additional states in order to get literal output. An edge is broken into smaller edges. For instance, the edge $0 \xrightarrow{011|E} 1$ is replaced by the path $0 \xrightarrow{0|-} E_1 \xrightarrow{1|-} E_2 \xrightarrow{1|E} 1$. Each state x_i stands for the prefix of length i of the encoding of x . The state x_i can be used for several edges, because the arrival state depends only on x . The transducer is given in Fig. 12.36.

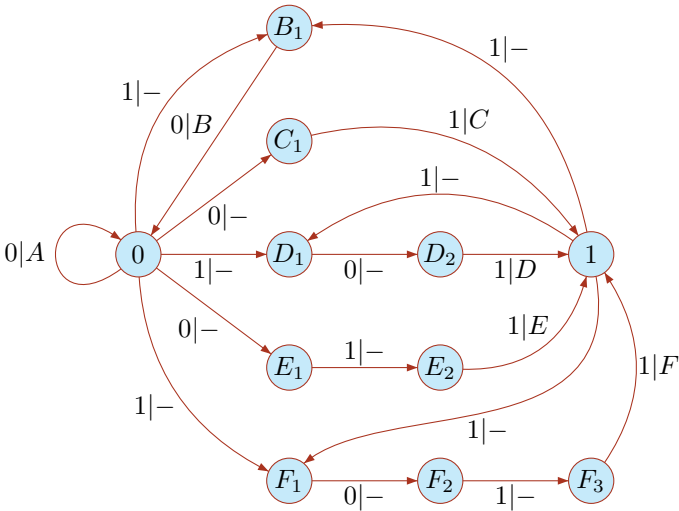


Fig. 12.36. Literal decoder for phonetization.

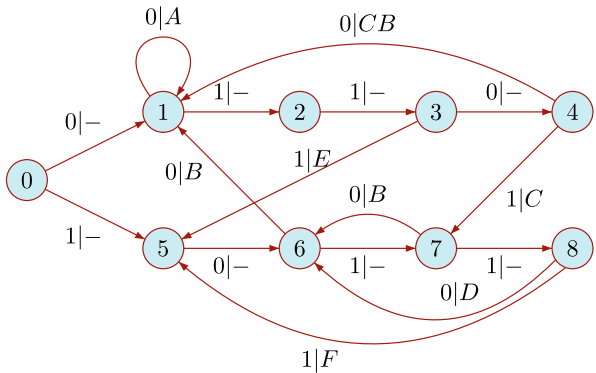


Fig. 12.37. A sequential transducer for the phonetization.

Table 12.4. The states of the transducer for the phonetization.

state	0	1	2	3	4	5	6	7	8
content	$\varepsilon, 0$	$A, 0$	A, B_1	C, B_1	$CB, 0$	ε, B_1	$B, 0$	B, B_1	D, D_1
		ε, C_1	$C, 1$	C, D_1	C, D_2	ε, D_1	ε, D_2	B, D_1	D, F_1
		ε, E_1	ε, E_2	C, F_1	C, F_2	ε, F_1	ε, F_2	B, F_1	D, B_1
			A, D_1	$E, 1$				$D, 1$	$F, 1$
			A, F_1					ε, F_3	
output		A	C	E	CB		B	D	F

It happens that the resulting transducer can be transformed into a sequential one. The result is shown in Fig. 12.37. The correspondence is given in Table 12.4. The value of the output function is given in the node.

The decoder is deterministic in its input and it outputs the decoded sequence with finite delay (this means that it uses a finite lookahead on the input). On termination of a correct input sequence, the last symbol to be produced is indicated by the corresponding state. This coding is used in [31] to build a syllabification algorithm, which produces for each word a parsing in syllables.

This kind of problem appears more generally in the framework of hyphenation in text processing software (see [16], for example), and also in “text-to-speech” synthesis [17].

The next example of codes for constrained sources is from [14]. It illustrates again how an ambiguous substitution can be converted into a UD encoding.

The three examples below have decoders, which are local (that is realizable by a sliding block decoder) for the first one, sequential for the second one, and simply unambiguous for the third one.

Example 12.23. Consider the constrained source on the symbols A, B, C with the constraint that B cannot follow A . We consider the substitution that maps A, B , and C to 0, 1, and 01, respectively. The mapping is realized by the transducer on the left-hand side of Fig. 12.38. It is easily seen that the constraint implies the unique decipherability property. Actually, a decoder can be computed by using the determinization algorithm. This yields the sequential decoder represented on the right-hand side of Fig. 12.38. It is even $(1, 0)$ -local since the last letter determines the state.

We now give a second, slightly more involved example. The source has four symbols A, B, C, D with the constraints of concatenation given by the matrix

$$M_2 = \begin{matrix} & A & B & C & D \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{matrix}.$$

We consider the encoding assigning 0, 1, 01, and 10 to A, B, C , and D , respectively. The decoder is given in Fig. 12.39. This transducer is sequential with output function indicated in the states. However, it is not local in input since there are two cycles labeled by 01.

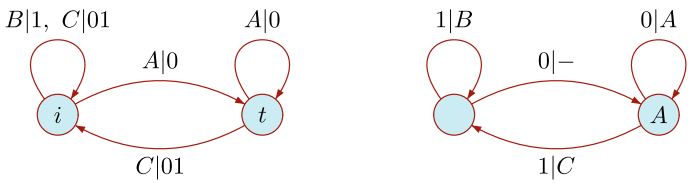


Fig. 12.38. An encoder and a local decoder.

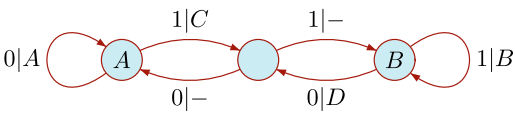


Fig. 12.39. A sequential decoder.

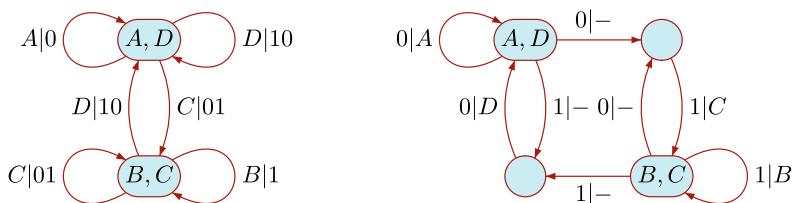


Fig. 12.40. An encoder and an unambiguous decoder.

The third example uses the same substitution but a different set of constraints, given by the matrix

$$M_3 = \begin{matrix} & A & B & C & D \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{matrix}.$$

The substitution is realized by the transducer shown in the left-hand side of Fig. 12.40. The transducer shown in the right-hand side is a decoder. It can be checked to be unambiguous. However, it is not equivalent to a sequential transducer because a sequence $0101\cdots$ has two potential decodings as $ADDD\cdots$ and as $CCCC\cdots$.

These examples are variable-length, and in fact variable-rate, codes. In applications where error propagation may be a problem, one can use fixed-rate block codes [38] or fixed-rate sliding-block codes [20]. The construction of the latter is dual to the state splitting method mentioned in the previous section.

The connection between variable-length codes, unambiguous automata, and local constraints has been further developed in [9, 52, 53].

12.11. Bifix Codes

Recall that a set of words C is a *suffix* code if no element of C is a proper suffix of another one. A set of words is called a *bifix* code if it is at the same time a prefix code and a suffix code. Bifix codes are also known as *reversible variable-length codes* (RVLC). The idea to study bifix codes goes back to [26, 59]. These papers already contain significant results. The first systematic study is in [60, 61]. The development of codes for video recording has renewed the interest in bifix codes [27, 69, 70].

One example is given by prefix codes which are equal to their reversal. The *reversal* of a word $w = a_1 \cdots a_n$, where a_1, \dots, a_n are symbols, is the word $\tilde{w} = a_n \cdots a_1$ obtained by reading w from right to left. The reversal of a set C , denoted \tilde{C} , is the set of reversals of its elements. For example, 01^*0 is a bifix code since it is prefix and equal to its reversal.

The use of bifix codes for transmission is linked with the possibility of limiting the consequences of errors occurring in the transmission using a bidirectional decoding scheme as follows. Assume that we use a binary bifix code to transmit data and that for the transmission, messages are grouped into blocks of N source symbols, encoded as N codewords. The block of N codewords is first decoded by using an ordinary left-to-right sequential decoding (Fig. 12.41). Suppose that the codewords x_1 up to x_{i-1} are correctly decoded, but that an error has occurred during transmission that makes it impossible to identify the next codeword in the rest of the message. Then a new decoding process is started, this time from right to left. If at most one error has occurred, then again the codewords from x_N down to x_{i+1} are decoded correctly. Thus, in a block of N encoded source symbols, at most one codeword will be read incorrectly.

These codes are used for the compression of moving pictures. Indeed, there are reversible codes with the same length distribution as the Golomb–Rice codes, as shown in [68]. The AVC standard mentioned previously recommends the use of these codes instead of the ordinary Golomb–Rice codes to obtain an error resilient coding (see [55]). The difference from the ordinary codes is that, in the base, the word 1^i0 is replaced by $10^{i-1}1$ for $i \geq 1$. Since the set of bases forms a bifix code, the set of all codewords is also a bifix code. Figure 12.42 represents the reversible Golomb–Rice codes of orders 0, 1, 2.

There is also a reversible version of the exponential Golomb codes, denoted by REG_k , which are bifix codes with the same length distribution. The code REG_0 is given by

$$REG_0 = \{0\} \cup 1\{00, 10\}^*\{0, 1\}1.$$

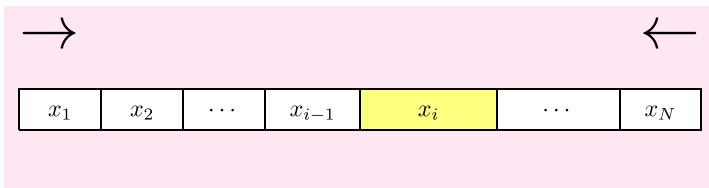


Fig. 12.41. The transmission of a block of codewords.

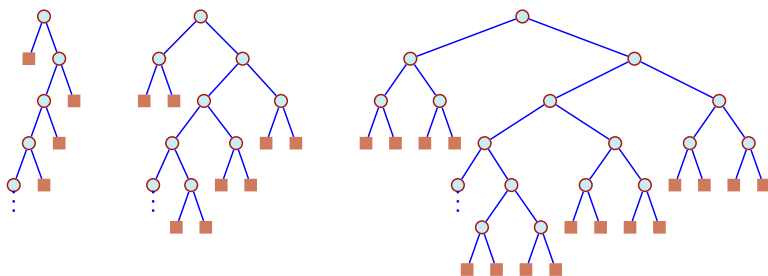


Fig. 12.42. The reversible Golomb–Rice codes of orders 0, 1, 2.

It is a bifix code because it is equal to its reversal. This comes from the fact that the set $\{00, 10\}^* \{0, 1\}$ is equal to its reversal because it is the set of words of odd length which have a 0 at each even position, starting at position 1.

The code of order k is

$$REG_k = REG_0\{0, 1\}^k.$$

The codes REG_k are represented for $k = 0$ and 2 in Fig. 12.43.

We now consider the length distribution of bifix codes. In contrast to the case of codes or of prefix codes, it is not true that any sequence $(u_n)_{n \geq 1}$ of non-negative integers such that $\sum_{n \geq 1} u_n k^{-n} \leq 1$ is the length distribution of a bifix code on k letters. For instance, there is no bifix code on the alphabet $\{a, b\}$, which has the same distribution as the prefix code $\{a, ba, bb\}$. Indeed, such a code must contain a letter, say a , and then the only possible word of length 2 is bb . On the other hand, the following result provides a sufficient condition for a length distribution to be realizable by a bifix code [4].

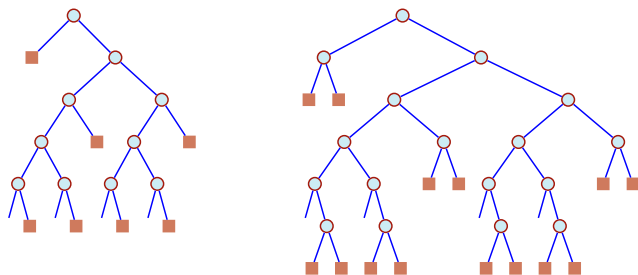


Fig. 12.43. The reversible exponential Golomb codes of orders 0 and 1.

Proposition 12.1. *For any sequence $(u_n)_{n \geq 1}$ of non-negative integers such that*

$$\sum_{n \geq 1} u_n k^{-n} \leq \frac{1}{2} \quad (12.17)$$

there exists a bifix code on an alphabet of k letters with length distribution $(u_n)_{n \geq 1}$.

Proof. We show by induction on $n \geq 1$ that there exists a bifix code X_n of length distribution $(u_i)_{1 \leq i \leq n}$ on an alphabet A of k symbols. It is true for $n = 1$ since $u_1 k^{-1} \leq 1/2$ and thus $u_1 < k$. Assume that the property is true for n . We have by Inequality (12.17)

$$\sum_{i=1}^{n+1} u_i k^{-i} \leq \frac{1}{2}$$

or equivalently, multiplying both sides by $2k^{n+1}$,

$$2(u_1 k^n + \cdots + u_n k + u_{n+1}) \leq k^{n+1}$$

whence

$$u_{n+1} \leq 2u_{n+1} \leq k^{n+1} - 2(u_1 k^n + \cdots + u_n k). \quad (12.18)$$

Since X_n is bifix by the induction hypothesis, we have

$$\text{Card}(X_n A^* \cap A^{n+1}) = \text{Card}(A^* X_n \cap A^{n+1}) = u_1 k^n + \cdots + u_n k.$$

Thus, we have

$$\begin{aligned} & \text{Card}((X_n A^* \cup A^* X_n) \cap A^{n+1}) \\ & \leq \text{Card}(X_n A^* \cap A^{n+1}) + \text{Card}(A^* X_n \cap A^{n+1}) \\ & \leq 2(u_1 k^n + \cdots + u_n k). \end{aligned}$$

It follows from Inequality (12.18) that

$$\begin{aligned} u_{n+1} & \leq k^{n+1} - 2(u_1 k^n + \cdots + u_n k) \\ & \leq \text{Card}(A^{n+1}) - \text{Card}((X_n A^* \cup A^* X_n) \cap A^{n+1}) \\ & = \text{Card}(A^{n+1} - (X_n A^* \cup A^* X_n)). \end{aligned}$$

This shows that we can choose a set Y of u_{n+1} words of length $n + 1$ on the alphabet A , which do not have a prefix or a suffix in X_n . Then $X_{n+1} = Y \cup X_n$ is bifix, which ends the proof. \square

Table 12.5. Maximal 2-realizable length distributions of length $N = 2, 3$, and 4.

N	2			3				4					
	u_1	u_2	$u(1/2)$	u_1	u_2	u_3	$u(1/2)$	u_1	u_2	u_3	u_4	$u(1/2)$	
	2	0	1.0000	2	0	0	1.0000	2	0	0	0	1.0000	
	1	1	0.7500	1	1	1	0.8750	1	1	1	1	0.9375	
				1	0	2	0.7500	1	0	2	1	0.8125	
								1	0	1	3	0.8125	
								1	0	0	4	0.7500	
	0	4	1.0000	0	4	0	1.0000	0	4	0	0	1.0000	
				0	3	1	0.8750	0	3	1	0	0.8750	
								0	3	0	1	0.8125	
				0	2	2	0.7500	0	2	2	2	0.8750	
								0	2	1	3	0.8125	
								0	2	0	4	0.7500	
				0	1	5	0.8750	0	1	5	1	0.9375	
								0	1	4	4	1.0000	
								0	1	3	5	0.9375	
								0	1	2	6	0.8750	
								0	1	1	7	0.8125	
								0	1	0	9	0.8125	
				0	0	8	1.0000	0	0	8	0	1.0000	
								0	0	7	1	0.9375	
								0	0	6	2	0.8750	
								0	0	5	4	0.8750	
								0	0	4	6	0.8750	
								0	0	3	8	0.8750	
								0	0	2	10	0.8750	
								0	0	1	13	0.9375	
								0	0	0	16	1.0000	

The bound $1/2$ in the statement of Proposition 12.1 is not the best possible. It is conjectured in [4] that the statement holds with $3/4$ instead of $1/2$. Some attempts to prove the conjecture have led to improvements over Proposition 12.1. For example, it is proved in [71] that $1/2$ can be replaced by $5/8$. Another approach to the conjecture is presented in [15].

For convenience, we call a sequence (u_n) of integers k -realizable if there is a bifix code on k symbols with this length distribution.

We fix $N \geq 1$ and we order sequences $(u_n)_{1 \leq n \leq N}$ of integers by setting $(u_n) \leq (v_n)$ if and only if $u_n \leq v_n$ for $1 \leq n \leq N$. If $(u_n) \leq (v_n)$ and (v_n) is k -realizable then so is (u_n) . We give in Table 12.5 the values of the maximal 2-realizable sequences for $N \leq 4$, with respect to this order. Set $u(z) = \sum_{n \geq 1} u_n z^n$. For each value of N , we list in decreasing lexicographic order the maximal realizable sequence with the corresponding value of the sum $u(1/2) = \sum u_n 2^{-n}$. The distributions with value 1 correspond to

Table 12.6. The length distributions of binary finite maximal bifix codes of degree at most 4.

d	1		2			3				4							
	2	1	0	4	1	0	0	8	1	0	0	0	16				1
										0	0	1	12	4			6
										0	0	2	8	8			6
										0	0	2	9	4	4		8
										0	0	3	5	8	4		6
										0	0	3	6	4	8		4
										0	0	3	6	5	4	4	4
										0	0	4	3	5	8	4	4
						0	1	4	4	2	0	1	0	5	12	4	2
											0	1	0	6	8	8	2
											0	1	0	6	9	4	4
											0	1	0	7	5	8	4
											0	1	0	7	6	5	4
											0	1	0	8	2	9	4
											0	1	1	3	9	8	4
											0	1	1	4	6	8	8
											0	1	1	4	6	9	4
											0	1	1	5	3	9	8
											0	1	2	2	4	9	12
		1			1				3								73

The same holds of course for the reversible one. The fact that the average length is an integer is a necessary condition for the existence of a reversible version of any regular prefix code, as we have already mentioned before. The average length of the Golomb code G_3 is easily seen to be $7/2$ for the uniform Bernoulli distribution. Since this is not an integer, there is no regular bifix code with the same length distribution $(0, 1, 3, 3, \dots)$. Actually, one may verify that there is not even a binary bifix code with length distribution $(0, 1, 3, 3, 2)$.

12.11.1. Exercises

Exercise 12.11.1. The aim of this exercise is to describe a method, due to Girod [28] (see also [58]), which allows a decoding in both directions for any finite binary prefix code. Let C be a finite binary prefix code and let L be the maximal length of the words of C . Consider a concatenation $c_1 c_2 \cdots c_n$ of codewords. Let

$$w = c_1 c_2 \cdots c_n 0^L \oplus 0^L \tilde{c}_1 \tilde{c}_2 \cdots \tilde{c}_n, \quad (12.19)$$

where \tilde{c} is the reverse of the word c and where \oplus denotes addition mod 2. Show that w can be decoded in both directions.

12.12. Synchronizing Words

A word v is said to be *synchronizing* for a prefix code C if for any words u, w , one has uvw in C^* only if uv and w are in C^* . Thus, the decoding of a message where v appears has to break at the end of v . A prefix code is said to be *synchronized* if there exists a synchronizing word. An occurrence of a synchronizing word limits the propagation of errors that have occurred during the transmission, as shown in the following example.

Example 12.24. Consider the prefix code $C = \{01, 10, 110, 111\}$. The word 110 is not synchronizing. Indeed, it appears in a non-trivial way in the parsing of **111|01**. On the contrary $v = 0110$ is synchronizing. Indeed, the only possible parsings of v in some context are $\dots 0|110|\dots$ and $\dots |01|10|\dots$. In both cases, the parsing has a cut point at the end of v . To see how an occurrence of v avoids error propagation, consider the message in the first row below and the corrupted version below it produced when an error has changed the third bit of the message from 0 to 1.

0	1		0	1		1	1		0	1		1		0	1		0	1
0	1		1	1		0	1		0	1		1		0	1		0	1

After the occurrence of 0110, the parsings on both rows become identical again. Thus, the occurrence of the word v has stopped the propagation of the error. Note that it can also happen that the resulting message does not have a decoding anymore. This happens, for example, if the second bit of the message is changed from 1 to 0. In this case, the error is detected and some appropriate action may be taken. In the previous case, the error is not detectable and may propagate for an arbitrary number of bits.

Example 12.25. The word 010 is synchronizing for the prefix code C used in the Franaszek code (see Fig. 12.30).

A synchronizing word can also be defined for a deterministic automaton. Let \mathcal{A} be a deterministic automaton. A word w is said to be *synchronizing* for \mathcal{A} if all paths labeled w end at the same state. Thus the state reached after reading a synchronizing word is independent of the starting state. A deterministic automaton is said to be *synchronized* if there exists a synchronizing word for the automaton.

An automaton is *strongly connected* if any pair of states is connected by some path. Let i be a state of a strongly connected deterministic automaton \mathcal{A} . Let C be the prefix code of first returns from i to i . Then C has

a synchronizing word if and only if \mathcal{A} is synchronized. First, let v be a synchronizing word for C . Then any path labeled v ends at state i . Indeed, if $p \xrightarrow{v} q$ let u, w be such that $i \xrightarrow{u} p$ and $q \xrightarrow{w} i$. Then uvw is in C^* , which implies that uv is in C^* . This implies that $q = i$ since the automaton is deterministic. Thus, v is synchronizing for the automaton \mathcal{A} . Conversely, if v is synchronizing for \mathcal{A} , then since \mathcal{A} is strongly connected, there is a word w such that all paths labeled vw end at state i . Then, vw is a synchronizing word for C .

Example 12.26. Let \mathcal{A} be the automaton represented in Fig. 12.45. Each word w defines an *action* on the set of states, which is the partial function that maps a state p to the state reached from p by the path labeled by the input word w . The set of first returns to state 1 is the maximal prefix code $C = \{00, 01, 110, 111\}$ of Example 12.24. In Table 12.7 the actions of words of length at most 4 are listed. The words are ordered by their length, and within words of the same length, by lexicographic order. Each column is reported only the first time it occurs. We stop at the first synchronizing word, which is 0110.

The road coloring theorem. There are prefix codes which are not synchronized. For example, if the lengths of the codewords of a non-empty prefix code are all multiples of some integer $p \geq 2$, then the code is not synchronized. The same observation holds for a strongly connected automaton. If the period of the underlying graph (i.e. the greatest common

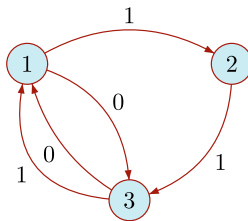


Fig. 12.45. A synchronized deterministic automaton.

Table 12.7. The action of words on the states of the automaton \mathcal{A} .

	0	1	00	01	10	11	001	011	100	101	110	111	0011	0110
1	3	2	1	1	—	3	2	2	—	—	1	1	3	1
2	—	3	—	—	1	1	—	—	3	2	3	2	—	—
3	1	1	3	2	3	2	1	3	1	1	—	3	2	1

divisor of the lengths of its cycles) is not 1, then the automaton is not synchronized. The *road coloring theorem* asserts the following.

Theorem 12.1. *Let G be a strongly connected graph with period 1 such that each vertex has two outgoing edges. Then there exists a binary labeling of the edges of G , which turns it into a synchronized deterministic automaton.*

The road coloring theorem was proposed as a conjecture by Adler et al. [3]. It was proved by Trahtman [65].

The name of this theorem comes from the following interpretation of a synchronizing word: if one assigns a color to each letter of the alphabet, the labeling of the edges of an automaton can be viewed as a coloring of the edges of the underlying graph. One may further identify the vertices of the graph with cities and the edges with roads connecting these cities. A synchronizing word then corresponds to a sequence of colors, which leads to a fixed city regardless of the starting point.

Example 12.27. Consider the automata represented on Fig. 12.46. These automata have the same underlying graph and differ only by the labeling. The automaton on the left is not synchronized. Indeed, the action of the letters on the subsets $\{1, 3\}$ and $\{2, 4\}$ exchanges these sets as shown in Fig. 12.47.

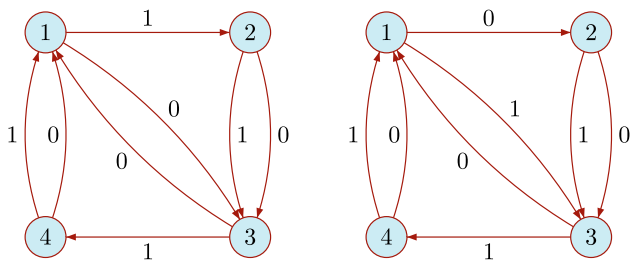


Fig. 12.46. Two different labelings: a non-synchronized and a synchronized automaton.

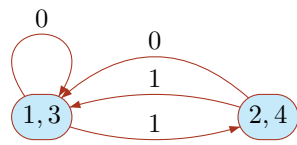


Fig. 12.47. The action on the sets $\{1, 3\}$ and $\{2, 4\}$.

On the other hand, the automaton on the right is synchronized. Indeed, 101 is a synchronizing word.

It is important to note that for equal letter costs, an optimal prefix code can always be chosen to be synchronized (provided the greatest common divisor of the lengths is 1). Indeed, the relabeling of an automaton accepting such a code does not change its length distribution. The Franaszek code of Sec. 12.9 is actually chosen in such a way that the code on the right-hand side of Fig. 12.30 is synchronized (010 is a synchronizing word).

For unequal letter costs, it has been proved that the result holds for finite maximal codes. Indeed, it is proved in [51] (see also [11]) that any finite maximal prefix code is commutatively equivalent to a synchronized prefix code.

12.13. Directions for Future Research

The field of variable-length codes and automata has a number of challenging mathematical open problems. They are often formulated as conjectures, some of which have been open for many years. Their solution would increase our understanding of these objects and give rise to new algorithms. Let us mention the following ones of particular importance.

The structure of finite maximal codes. It is not known whether it is decidable whether a finite code is or is not contained in a finite maximal code. In contrast, it is known that any finite code is contained in a regular maximal code (see [11]).

Another open question is the status of the conjecture concerning the commutative equivalence of any finite maximal code to a prefix code. This conjecture would itself be solved if one could prove the factorization conjecture asserting that for any finite maximal code on the alphabet A , there exist two finite sets of words P, Q on the alphabet A such that any word on the alphabet A has a unique expression of the form $pc_1c_2 \cdots c_nq$ with $p \in P$, $c_i \in C$, and $q \in Q$.

Optimal prefix codes. There is still some work to be done to derive efficient methods for building an optimal prefix code corresponding to a source with an infinite number of elements.

Synchronized automata. It is conjectured that for any synchronized deterministic automaton with n states, there exists a synchronizing word of length at most $(n-1)^2$. This conjecture is known as Černý's conjecture. The length of the shortest synchronizing word has practical significance

since an occurrence of a synchronizing word has an error-correcting effect. The best known upper bound is cubic. The conjecture is known to be true in several particular cases.

Constrained channels. For a given constrained channel and allowable code rate, the problem of designing codes that achieve the minimum possible number of encoder states or the minimum sliding block decoder window size remains open.

Bifix codes. No method similar to the Huffman algorithm is known for building an optimal bifix code given a source with weights. It is perhaps related to the fact that the length distributions of bifix codes are not well understood. In particular, the question whether any sequence $(u_n)_{n \geq 1}$ such that $\sum_{n \geq 1} u_n k^{-n} \leq 3/4$ is the length distribution of a bifix code over a k -letter alphabet is still open.

12.14. Conclusion

We have described basic properties of variable-length codes, and some of their uses in the design of optimal codes under various constraints. We have shown some of the relations between codes, finite state automata and finite transducers, both devices for encoding and decoding.

Optimal codes. Among variable-length codes, the most frequently used are prefix codes. These codes are instantaneously decipherable codes. The Elias, Golomb, and Golomb–Rice codes are examples of prefix codes that are infinite, and that encode non-negative integers by an algorithm that is easy to implement. Other prefix codes that encode integers have been designed. For a systematic description, see [58]. Optimal prefix codes for various constraints have been given in this chapter. It is interesting to note that the general problem is still open, and the research is still going on (see [29] for a recent contribution).

General codes. Prefix codes are special codes. Other families of codes have been studied, and a general theory of variable-length codes addresses the properties of these codes. In this context, it is natural to associate with any code C an automaton that recognizes the set C^* of all sequences of codewords. Many properties or parameters of codes are reflected by features of these automata.

It appears that properties of codes are combinatorial in nature; however, they can also be described in an algebraic way based upon

the structural properties of the associated automaton or the algebraic properties of the transition monoid of the automaton, also called the syntactic monoid. See [11].

Constrained channels. We have also briefly investigated constraints on the channel. The theory of sliding block codes is much broader in scope than we could convey here. Research is motivated by applications, as we have illustrated by the example of the Franaszek code. For more information along these lines, see [7, 45].

12.15. Solutions to Exercises

Solution to 12.6.1. One has $l_i \geq \log 1/p_i$ and thus

$$\sum_i 2^{-l_i} \leq \sum_i 2^{-\log \frac{1}{p_i}} = \sum_i 2^{\log p_i} = \sum_i p_i = 1.$$

Let C be a prefix code with length distribution ℓ_i . Since $\ell_i \leq \log 1/p_i + 1$, its average length W satisfies

$$W = \sum p_i \ell_i \leq \sum p_i \left(\log \frac{1}{p_i} + 1 \right) = p_i \log \frac{1}{p_i} + 1 = H + 1.$$

Solution to 12.6.2. Each word of P followed by a letter is either in P or in C , but not in both. Moreover, any non-empty word in C or P is obtained in this way. This shows that $PA \cup \{\epsilon\} = P \cup C$. Counting the elements on both sides gives $\text{Card}(P) \text{Card}(A) + 1 = \text{Card}(P) + \text{Card}(C)$.

Solution to 12.6.3. Let P be the set of words on the alphabet A which do not have s as a factor. Then P is also the set of proper prefixes of the words of the maximal prefix code X . Thus, we have $P\{0, 1\} \cup \{\epsilon\} = X \cup P$, whence $f_P(z)(1 - 2z) = 1 - f_X(z)$. On the other hand, we have $Ps = XQ$ and thus $z^p f_P(z) = f_X(z) f_Q(z)$. Combining these relations and solving for f_X gives the desired solution.

Solution to 12.6.4. One has $Q = \{\epsilon, 01\}$ and thus $f_Q(z) = 1 + z^2$.

Solution to 12.6.5. Since $z^p + 2zf_X(z) = f_X(z) + f_U(z)$, the result follows from (12.12). Prefix synchronized codes were introduced by Gilbert [24], who conjectured that for any $n \geq 1$, the maximal cardinality is obtained for an *unbordered word* such as $11 \cdots 10$ (a word is called unbordered if no non-empty prefix is also a suffix). This conjecture was solved positively

by Guibas and Odlyzko [32] who also showed that the generalization to alphabets with k symbols is true for $k \leq 4$ but false for $k \geq 5$.

Solution to 12.6.6. Consider the cost on B defined by $\text{cost}(b) = -\log \pi(b)$. Then $\text{cost}(c) = -\log \pi(c)$ for each codeword and thus a code with minimal cost (with equal weights) will have maximal average length with respect to π .

Solution to 12.7.1. One has, using the fact that $\sum_{n \geq 0} np^n q = p/q$,

$$\begin{aligned} H &= - \sum_{n \geq 0} p^n q \log(p^n q) = - \sum_{n \geq 0} p^n q \log p^n - \sum_{n \geq 0} p^n q \log q \\ &= - \sum_{n \geq 0} np^n q \log p - \log q = -p/q \log p - \log q, \end{aligned}$$

whence the result. If $p^{2^k} = 1/2$ then $p = 2^{-2^{-k}}$. For $x \geq 0$, we have $2^{-x} \geq 1 - x \ln 2 \geq 1 - x$. Thus

$$q = 1 - p = 1 - 2^{-2^{-k}} \leq 2^{-k}.$$

Taking the logarithm of both sides gives $\log(1/q) \geq k$. On the other hand, since $\log x \geq x - 1$ for $1 \leq x \leq 2$, we have $p/q \log(1/p) \geq 1$. Combining these inequalities, we obtain $H \geq k + 1$, whence $H + 1 \geq k + 2 = \lambda_{GR_k}$.

Solution to 12.7.2. We use an induction on m . The property clearly holds for $m = 3$. Let $m \geq 4$ and set $w_0 = w_{m-1} + w_m$. The sequence $w_0 \geq w_1 \geq \dots \geq w_{m-2}$ is quasi-uniform since $w_{m-3} + w_{m-2} \geq w_{m-1} + w_m$. By the induction hypothesis, an optimal binary tree for the sequence $w_0 \geq w_1 \geq \dots \geq w_{m-2}$ has leaves of heights k and possibly $k + 1$ for some $k \geq 1$. The leaf corresponding to w_0 has to be at height k since it has maximal weight. We replace it by two leaves of level $k + 1$ with weights w_{m-1} and w_m . The result is clearly an optimal binary tree with leaves at height k and $k + 1$. This argument appears in [22].

Solution to 12.8.1. Suppose that $c_1 c_2 \dots c_n = c'_1 c'_2 \dots c'_m$. We may suppose that the length of the word $c_2 \dots c_n$ is larger than d , padding both sides on the right by an appropriate number of words c from C . Then, the condition implies that $c_1 = c_2$, and so on.

Solution to 12.8.2. The set C is the code of Example 12.4. It is not weakly prefix since for any n , $(10)(00)^n$ is a prefix of $(100)(00)^n$.

The code C' is weakly prefix. Indeed, the decoding of a word beginning with $001 \dots$ has to start with 001 .

Solution to 12.8.3. Suppose first that C has a sequential decoder. Let d be the maximal length of the values of the output function. Then C is weakly prefix with delay dL , where L is the maximal length of the codewords.

Conversely, if C is a weakly prefix code, the determinization algorithm gives a sequential decoder.

Solution to 12.9.1. First, observe that D , viewed as a code over the alphabet C , is a prefix code. Thus D is a code. Assume now that u, v are such that uv and vu are in D^* . Then, since C is circular, u and v are in C^* . Set $u = x_1 \cdots x_n$ and $v = x_{n+1} \cdots x_{n+m}$. Since uv is in D^* , we have $x_{n+m} \neq x$ and thus v is in D^* . Similarly, since vu is in D^* , we have $x_n \neq x$ and thus u is in D^* . Thus D is circular.

Solution to 12.9.2. We use repeatedly Exercise 12.9.1 to form a sequence C_0, C_1, \dots of circular codes. To form C_{i+1} , we choose one element x_i in C_i and select words of the form $x_i^k y$ with $y \neq x_i$. We start with $C_0 = \{a, b, c\}$, which is obviously circular, and we choose $x_0 = c$. Then $C_1 = \{a, b, ca, cb\}$ is a circular code. Choosing next $x_1 = cb$, we build $C_2 = \{a, b, ca, cba\}$. Then, we choose $x_2 = b$ to obtain $C_3 = \{a, ba, ca, cba\}$. A final choice of $x_3 = a$ shows that C is of the required form.

Note: this construction is used in a more general setting in the study of circular codes [10].

Solution to 12.11.1. The definition of w being symmetrical, it is enough to show that w can be decoded from left to right. By construction, c_1 is a prefix of w and the first codeword can therefore be decoded. However, this also identifies the prefix of length $L + |c_1|$ of the second term of the right-hand side of (12.19). Adding this prefix to the corresponding prefix of w gives a word beginning with $c_1 c_2$ and thus identifies c_2 , and so on.

12.16. Questions and Answers

Question 12.1. Compute the entropy of the source consisting of the five most frequent words in English with the normalized weights given in Table 12.8. Compare the result with the average length of the prefix code of Example 12.1.

Answer. One obtains $H = 2.197$. The average length is $\lambda = 2.29$. Thus, the average length is slightly larger than the entropy.

Table 12.8. Normalized weights of the five most frequent English words.

A	AND	OF	THE	TO
0.116	0.174	0.223	0.356	0.131

Question 12.2. What is the result of the Huffman algorithm applied to the five most frequent words in English with the frequencies given in Question 12.1? What is the average length?

Answer. The result is represented in Fig. 12.48. The average length in this case is $\lambda = 2.247$ (which is smaller than for the code of Example 12.1).

Question 12.3. What is the result of the Garsia–Wachs algorithm applied to the coding of the five most frequent words in English with the distribution given in Question 12.1?

Answer. The result of the combination step is the tree of Fig. 12.49. The recombination step gives the tree of Example 12.1.

Question 12.4. Consider the source having six symbols of costs $3, 4, \dots, 8$ respectively. Show that the capacity C of this source satisfies $C > 1/2$.

Answer. One has $C = \log(1/\rho)$ where ρ is the positive root of $f(z) = 1$ with $f(z) = z^3 + z^4 + z^5 + z^6 + z^7 + z^8$. Since $f(z) = z^2(z + z^2)(1 + z^2 + z^4)$, we have $f(\sqrt{2}/2) > 0.5 \times 1.2 \times 1.75 > 1$. This shows that $\rho < \sqrt{2}/2$ and thus $C > 1/2$.

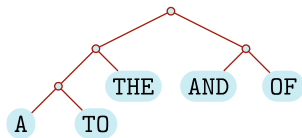


Fig. 12.48. The result of Huffman's algorithm.

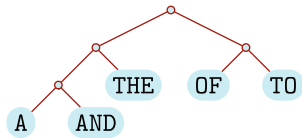


Fig. 12.49. The result of the Garsia–Wachs algorithm.

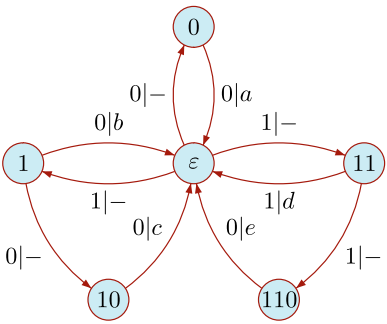


Fig. 12.50. An unambiguous decoder.

Question 12.5. What is the value of m corresponding to (12.14) for the geometric distribution such that $p = 0.95$? What values of p correspond to $m = 8$?

Answer. Since $-(\log(1 + p)/\log p) = 13.35$, we obtain $m = 14$. For $m = 8$, we have $9.11 < p < 9.22$.

Question 12.6. Write the encoding of the first 10 integers for the Golomb code G_4 .

Answer. The following table is obtained:

n	codeword
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111
8	1000
9	1001

Question 12.7. Is the encoding of a, b, c, d, e by 00, 10, 100, 11, 110 UD? Compute a decoder for this encoding.

Answer. There are only two possibilities to obtain two factorizations starting at the same point. Both alternate indefinitely and never come to

Table 12.9.

name		output
1	(ε, e)	ε
2	$(a, e), (\varepsilon, 0)$	a
3	$(aa, e), (a, 0), (\varepsilon, 00)$	aa

coincide. Thus, the code is UD.

$${}_2^110^10_20^10_20^1, \quad {}_2^111^10_20^10_20^1.$$

The method described in Sec. 12.8 gives the unambiguous transducer of Fig. 12.50.

Question 12.8. Design a sequential transducer decoding the weakly prefix code $C = \{0, 001\}$.

Answer. The general method gives the unambiguous transducer of Fig. 12.51 with $a = 0$ and $b = 001$. The procedure described in Sec. 12.8 applied to the transducer of Fig. 12.51 gives the sequential transducer of Fig. 12.52. We obtain three states given in Table 12.9 with the values of the output function in the last column.

Question 12.9. What is the labeled graph corresponding to the $[1,7]$ -constraint?

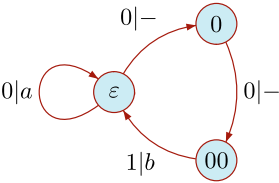


Fig. 12.51. An unambiguous decoder.

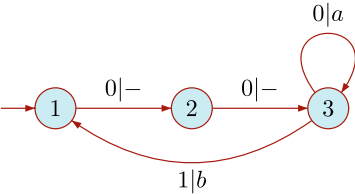
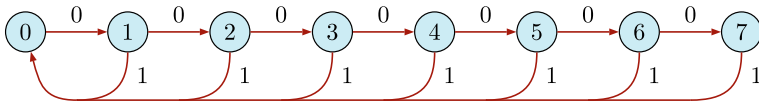
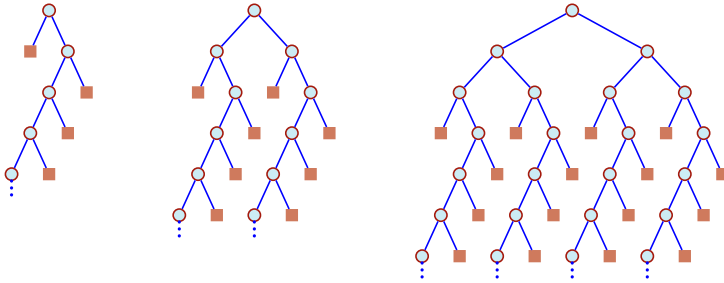


Fig. 12.52. A sequential transducer.

Fig. 12.53. The $[1,7]$ constraint.Fig. 12.54. The reversals of the codes RGR_k , for $k = 0, 1, 2$.

Answer. The graph is the same as for the $[2, 7]$ -constraint with one more arrow to allow the block 101. See Fig. 12.53.

Question 12.10. What is the reversal of the reversible Golomb–Rice code of order 1?

Answer. The expression of the reversal of RGR_k is $\widetilde{RGR}_k = RGR_0\{0, 1\}^k$ since RGR_0 is its own reversal. The corresponding trees are represented in Fig. 12.54.

12.17. Keywords

The following list of keywords is a kind of dictionary of terms used in this chapter with an explanation of each entry.

Finite automaton

A finite set of states, together with two distinguished subsets called the sets of initial and terminal states, and a set of edges which are triples consisting of a pair of states and a symbol in some (finite) alphabet A .

Unambiguous automaton

A finite automaton such that, for every pair of states and every word over the alphabet, there is at most one path (i.e. sequence of edges) from the first state of the pair to the second that is labeled with the word.

Deterministic automaton

A finite automaton with a unique initial state such that, for each state and each symbol in an alphabet, there is at most one edge starting in that state and labeled with that symbol.

Determinization algorithm

An algorithm that transforms any finite automaton into an equivalent deterministic finite automaton.

Regular set

A set of words generated by paths in a finite automaton, starting in an initial state and ending in a terminal state.

Transducer

A finite state automaton with output realizing a relation between words on an input alphabet A and words on an output alphabet B . It is similar to an automaton, but edges are quadruples consisting of a pair of states, a word over A , and a word over B . The main purpose for transducers is decoding. In this case, A is the channel alphabet and B is the source alphabet.

Literal transducer

A transducer such that each input label is a single letter.

Unambiguous transducer

A literal transducer whose associated input automaton is unambiguous.

Deterministic transducer

A literal transducer whose associated input automaton is deterministic.

Sequential transducer

A deterministic transducer and an output function. This function maps the terminal states of the transducer into words on the output alphabet. The function realized by a sequential transducer is obtained by appending to the value of the deterministic transducer the image of the output function on the terminal state.

Alphabetic coding

An order preserving encoding. The source and the channel alphabets are ordered, and the codewords are ordered lexicographically.

Optimal source coding

An encoding that minimizes the weighted cost. The weights are on the source alphabet and the costs on the channel alphabet.

Code

A set of non-empty words over an alphabet A such that every concatenation of codewords is uniquely decipherable.

Prefix (suffix) code

A set of non-empty words over an alphabet A such that no word in the set is a proper prefix (suffix) of another one.

Reversible code

A code which is prefix and which is also prefix when the words are read from right to left. Such a code is also called bifix.

Maximal code

A code (the image of an encoding) that is not strictly contained in another code.

Constrained channel

A communication channel that imposes constraints on the channel symbol sequences that can be transmitted.

Constrained source

A source that imposes constraints on the sequences of source alphabet symbols that it generates.

Symbolic dynamics

The study of symbolic representations of dynamical systems and code mappings between such representations.

Sliding block decoder

A decoder that operates on strings of channel symbols with a window of fixed size. The decoder uses m symbols before the current one and a symbols after it (m is for memory and a for anticipation). According to the value of the symbols between time $n - m$ and time $n + a$, the value of the n th source symbol is determined.

State splitting algorithm

An algorithm that transforms a finite-state graph representation of a constraint into a sliding-block decodable finite-state encoder.

Commutative equivalence

A relation between two codes in which there is a one-to-one correspondence between codewords such that corresponding pairs of words have the same number of occurrences of each alphabet symbol (that is, they are anagrams).

Channel capacity

The maximum amount of information that can be transmitted over the channel per unit cost. In many applications, the cost of a symbol corresponds to the number of channel uses or the time required to transmit it. The capacity is then a measure of the amount of information that can be transmitted over the channel per channel use.

Binary source entropy

A measure of the average number of bits per symbol required to represent a source with weighted source symbols.

Generating series

The power series whose coefficients are the number of words or the probabilities of words of each length.

Synchronizing word

A word for a finite automaton such that all paths labeled by this word end in the same state.

Geometric distribution

A probability distribution π on the set of non-negative integers such that $\pi(n+1)/\pi(n)$ is constant.

Acknowledgments

The authors would like to thank Frédérique Bassino, Julien Clément, Éric Incerti, Claire Kenyon, Éric Laporte, and Olivier Vénard for their help in the preparation of this text, and the anonymous referees for their helpful comments.

References

1. R. L. Adler, D. Coppersmith and M. Hassner, Algorithms for sliding block codes, *IEEE Trans. Inform. Theory* **29**, 1 (1983), 5–22.
2. R. L. Adler, J. Friedman, B. Kitchens and B. H. Marcus, State splitting for variable-length graphs, *IEEE Trans. Inform. Theory* **32**, 1 (1986), 108–113.
3. R. L. Adler, L. W. Goodwyn and B. Weiss, Equivalence of topological Markov shifts, *Israel J. Math.* **27**, 1 (1977), 48–63.
4. R. Ahlswede, B. Balkenhol and L. H. Khachatryan, Some properties of fix-free codes, *Proc. 1st Int. Seminar on Coding Theory and Combinatorics, Thakadzor, Armenia* (1996), pp. 20–33.
5. R. B. Ash, *Information Theory*, Dover Publications Inc., New York (1990), Corrected reprint of the 1965 original.
6. M.-P. Béal, The method of poles: A coding method for constrained channels, *IEEE Trans. Inform. Theory* **36**, 4 (1990), 763–772.
7. M.-P. Béal, *Codage symbolique*, Masson (1993).
8. M.-P. Béal, Extensions of the method of poles for code construction, *IEEE Trans. Inform. Theory* **49**, 6 (2003), 1516–1523.
9. M.-P. Béal and D. Perrin, Codes, unambiguous automata and sofic systems, *Theor. Comput. Sci.* **356**, 1–2 (2006), 6–13.
10. J. Berstel and D. Perrin, *Theory of Codes*, Academic Press (1985).
11. J. Berstel, D. Perrin and C. Reutenauer, *Codes and Automata*, Cambridge University Press (2009).
12. J. Berstel and C. Reutenauer, *Rational Series and their Languages*, Springer-Verlag (1988).
13. V. Bruyère and M. Latteux, Variable-length maximal codes, *Automata, Languages and Programming*, (Paderborn, 1996), *Lecture Notes in Computer Science*, Vol. 1099, Springer-Verlag (1996) pp. 24–47.
14. M. Dalai and R. Leonardi, Non prefix-free codes for constrained sequences, *IEEE Int. Symposium on Information Theory* (2005), pp. 1534–1538.
15. C. Deppe and H. Schnettler, On q -ary fix-free codes and directed deBruijn graphs, *IEEE Int. Symposium on Information Theory* 2(006), pp. 1482–1485.
16. J. Désarménien, La division par ordinateur des mots français: Application à \TeX , *Tech. Sci. Informa.* **5**, 4 (1986), 251–265.
17. T. Dutoit, *An Introduction to Text-To-Speech Synthesis*, Kluwer (1997).
18. S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press (1974).
19. P. Elias, Universal codeword sets and representations of the integers, *IEEE Trans. Inform. Theory* **21**, 2 (1975), 194–203.
20. J. L. Fan, B. H. Marcus and R. M. Roth, Lossless sliding-block compression of constrained systems, *IEEE Trans. Inform. Theory* **46**, 2 (2000), 624–633.
21. P. A. Franaszek, Run-length-limited variable length coding with error propagation limitation, US Patent 3,689,899, 1972.
22. R. G. Gallager and D. C. van Voorhis, Optimal source codes for geometrically distributed integer alphabets, *IEEE Trans. Inform. Theory* **21**, 2 (1975), 228–230.

23. A. M. Garsia and M. L. Wachs, A new algorithm for minimum cost binary trees, *SIAM J. Comput.* **6**, 4 (1977), 622–642.
24. E. N. Gilbert, Synchronization of binary messages, *IRE Trans. Inform. Theory* **6** (1960), 470–477.
25. E. N. Gilbert, Coding with digits of unequal cost, *IEEE Trans. Inform. Theory* **41**, 2 (1995), 596–600.
26. E. N. Gilbert and E. F. Moore, Variable length binary encodings, *Bell Sys. Tech. J.* **38** (1959), 933–967.
27. D. Gillman and R. Rivest, Complete variable length fix-free codes, *Design. Codes Cryptogr.* **5** (1995), 109–114.
28. B. Girod, Bidirectionally decodable streams of prefix code words, *IEEE Commun. Lett.* **3**, 8 (1999), 245–247.
29. M. J. Golin, C. Kenyon and N. E. Young, Huffman coding with unequal letter costs, *ACM Symposium Theory on Computing* (2002), pp. 785–791.
30. S. W. Golomb, Run-length encodings, *IEEE Trans. Inform. Theory*, **12**, 3 (1966), 399–401.
31. G. Gönenç, Unique decipherability of codes with constraints with application to syllabification of Turkish words, *COLING 1973: Computational and Mathematical Linguistics: Proceedings of the International Conference on Computational Linguistics*, Vol. 1, Firenze, Italy (1973), pp. 183–193.
32. L. J. Guibas and A. M. Odlyzko, Maximal prefix-synchronized codes, *SIAM J. Appl. Math.* **35**, 2 (1978), 401–418.
33. C. D. Heegard, B. H. Marcus and P. H. Siegel, Variable-length state splitting with applications to average runlength-constrained (ARC) codes, *IEEE Trans. Inform. Theory* **37**, 3, Part 2 (1991), 759–777.
34. T. C. Hu and M.-T. Shing, *Combinatorial Algorithms*, 2nd edn. Dover Publications Inc., Mineola, NY, (2002).
35. T. C. Hu and A. C. Tucker, Optimal computer search trees and variable-length alphabetical codes, *SIAM J. Appl. Math.* **21** (1971), 514–532.
36. T. C. Hu and P. A. Tucker, Optimal alphabetic trees for binary search, *Inform. Process. Lett.* **67**, 3 (1998), 137–140.
37. D. A. Huffman, A method for the construction of minimum redundancy codes, *Proceedings of the Institute of Electronics and Radio Engineers*, Vol. 40, No. 10, September 1952, pp. 1098–1101.
38. K. A. S. Immink, A practical method for approaching the channel capacity of constrained channels, *IEEE Trans. Inform. Theory* **43**, 5 (1997), 1389–1399.
39. K. A. S. Immink, *Codes for Mass Data Storage Systems*, 2nd edn. Shannon Foundation Publishers (2004).
40. A. Itai, Optimal alphabetic trees, *SIAM J. Comput.* **5**, 1 (1976), 9–18.
41. R. M. Karp, Minimum redundancy codes for the discrete noiseless channel, *IRE Trans. Inform. Theory* **7** (1961), 27–38.
42. J. H. Kingston, A new proof of the Garsia-Wachs algorithm, *J. Algorithms* **9**, 1 (1988), 129–136.
43. D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*, 2nd edn. Addison-Wesley (1998).

44. Z. Kohavi, *Switching and Automata Theory*, 2nd edn. McGraw-Hill (1978).
45. D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding* Cambridge University Press, Cambridge (1995).
46. M. Lothaire, *Applied Combinatorics on Words*, Encyclopedia of Mathematics and its Applications, Vol. 105, Cambridge University Press, Cambridge (2005).
47. B. H. Marcus, R. M. Roth and P. H. Siegel, Constrained systems and coding for recording channels, *Handbook of Coding Theory*, eds. V. S. Pless and W. C. Huffman, Elsevier (1998).
48. R. J. McEliece, *The Theory of Information and Coding*, student edn. Encyclopedia of Mathematics and its Applications, Vol. 86, Cambridge University Press, Cambridge (2004), with a foreword by Mark Kac.
49. K. Mehlhorn, An efficient algorithm for constructing nearly optimal prefix codes, *IEEE Trans. Inform. Theory* **26**, 5 (1980), 513–517.
50. N. Merhav, G. Seroussi and M. J. Weinberger, Optimal prefix codes for sources with two-sided geometric distributions, *IEEE Trans. Inform. Theory* **46**, 1 (2000), 121–135.
51. D. Perrin and M.-P. Schützenberger, Synchronizing prefix codes and automata and the road coloring problem, *Symbolic Dynamics and its Applications* (New Haven, CT, 1991), Contemporary Mathematics, Vol. 135, American Mathematical Society Providence, RI (1992), pp. 295–318.
52. A. Restivo, Codes and local constraints, *Theoret. Comput. Sci.* **72**, 1 (1990), 55–64.
53. C. Reutenauer, Ensembles libres de chemins dans un graphe, *Bull. Soc. Math. France* **114**, 2 (1986), 135–152.
54. R. F. Rice, Some practical universal noiseless coding techniques, Technical Report, Jet Propulsion Laboratory (1979).
55. I. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*, Wiley (2003).
56. E. Roche and Y. Schabes, eds., *Finite-State Language Processing*, MIT Press (1997).
57. J. Sakarovitch, *Elements of Automata Theory*, Cambridge University Press (2009).
58. D. Salomon, *Variable-Length Codes for Data Compression*, Springer-Verlag (2007).
59. M.-P. Schützenberger, On an application of semigroup methods to some problems in coding, *IEE Trans. Inform. Theory* **2** (1956), 47–60.
60. M.-P. Schützenberger, On a family of submonoids, *Publ. Math. Inst. Hungar. Acad. Sci. Ser. A* **VI** (1961), 381–391.
61. M.-P. Schützenberger, On a special class of recurrent events, *Ann. Math. Statist.* **32** (1961), 1201–1213.
62. C. E. Shannon, A mathematical theory of communication, *Bell Sys. Tech. J.* **27** (1948), 379–423, 623–656.
63. P. W. Shor, A counterexample to the triangle conjecture, *J. Combin. Theory Ser. A* **38** (1983), 110–112.

64. J. Teuhola, A compression method for clustered bit-vectors, *Inf. Process. Lett.* **7**, 6 (1978), 308–311.
65. A. N. Trahtman, The road coloring problem, *Israel J. Math.* (2008), to appear.
66. B. P. Tunstall, *Synthesis of noiseless compression codes*. PhD Thesis, Georgia Institute of Technology (1967).
67. B. Varn, Optimal variable length codes (arbitrary symbol cost and equal code word probability), *Inform. Contr.* **19** (1971), 289–301.
68. J. Wen and J. Villasenor, Reversible variable length codes for efficient and robust image and video coding, *IEEE Data Compression Conference* (1998), pp. 471–480.
69. M. W. Y. Takishima and H. Murakami, Reversible variable length codes, *IEEE Trans. Commun.* **43**, 2/3/4 (1995), 158–162.
70. C. Ye and R. W. Yeung, Some basic properties of fix-free codes, *IEEE Trans. Inform. Theory* **47**, 1 (2001), 72–87.
71. S. Yekhanin, Improved upper bound for the redundancy of fix-free codes, *IEEE Trans. Inform. Theory* **50**, 11 (2004), 2815–2818.