FOUNDATIONS OF CODING

Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory

Jiří Adámek
Czech Technical University in Prague

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
Chichester • New York • Brisbane • Toronto • Singapore

This page intentionally left blank

FOUNDATIONS OF CODING

This page intentionally left blank

FOUNDATIONS OF CODING

Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory

Jiří Adámek
Czech Technical University in Prague

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
Chichester • New York • Brisbane • Toronto • Singapore

Library of Congress Cataloging in Publication Data:

Adámek, Jiří, ing.

Foundations of coding: theory and applications of error-correcting codes, with an introduction to cryptography and information theory / Jiří Adámek.

p. cm.

"A Wiley-Interscience publication."
Includes bibliographical references and index.

ISBN 0-471-62187-0
1. Coding theory. I. Title.

QA268.A36 1991
003'.54 – dc20 90-20905
CIP
Rev.

10 9 8 7 6 5 4

To Honza, Kuba, Jirka, and Ondřej

This page intentionally left blank

Preface

Coding theory is a fascinating field combining elegant mathematical theories with constructions of a major practical impact.

This book is devoted to constructions of

(1) error-correcting codes,

(2) secrecy codes, and

(3) codes used in data compression.

The stress is on the first direction: we introduce a number of important classes of error-detecting and error-correcting codes, and we present their decoding methods. Some of these constructions require a deep background in modern algebra, and we carefully provide such background. Secret codes are treated only briefly; we mainly explain the role of error-correcting codes in modern cryptography. Data compression and other topics related to information theory are briefly discussed in the first part of the book.

The material is presented in a way making it possible to appreciate both the beauty of the theory and the scope of practical applications. We use the definition-theorem-proof style usual in mathematical texts (since the reader can thus skip a proof to keep continuity of the text and return to it later), but formalism is avoided as much as possible.

The book evolved from a series of lectures I held at the Czech Technical University in Prague in 1985–1990. They were based primarily on the following excellent textbooks which the reader may use for further reading: *Information Theory and Coding,* Abramson (1963),* *Theory and Practise of Error Control Codes,* Blahut (1983), *The Theory of Error-Correcting Codes,* MacWilliams and Sloane (1981), and *An Introduction to Cryptology,* van Tilborg (1988).

Jiří Adámek

*A name followed by a year in parentheses refers to the list of references at the end of the book.

This page intentionally left blank

Contents

This page intentionally left blank

FOUNDATIONS OF CODING

This page intentionally left blank

Introduction

Data transmision and data storage suffer from errors created by noise. Techniques for combatting noise have been used for a long time. They range from simple ones, e.g., adding a parity check symbol to every byte, to modern complex error-correcting techniques described in this book. The basic idea of error correction by a block code (i.e., a code in which all code words have the same length) is simple: code words must be "wide apart" from each other. That is, two distinct code words have a large Hamming distance, which means the number of symbols in which the words differ. Then the code corrects errors as follows: the word received is corrected to the nearest code word (in the sense of the Hamming distance). If the number of errors created by noise is smaller than one-half of the minimum Hamming distance of code words, then the correction is well done. Thus, the theory of error-correcting block codes is concerned with a construction of "good" codes with large Hamming distances. "Good" means that (1) the number of code words is as high as possible (to keep the redundancy low) and (2) an efficient technique for error correction is known (to make the search for the nearest code word fast).

Besides block codes, there is another class of error-correcting codes, called convolutional codes, in which memory plays a role: the message is again divided into blocks, but each block sent depends on a certain number of preceding blocks. The theory of convolutional codes is less rich than that of block codes: whereas good convolutional codes have been found by computer search, good block codes result from the algebraic theory presented in this book. However, the importance of convolutional codes in practical applications is ever increasing.

The theory of error-correcting codes is closely related to the theory of information, and the first part of this book is devoted to the foundations of information theory. Both of these theories were initiated by the pioneering paper of Claude Shannon (1948) in which he introduced entropy as a measure of information contained in an average symbol of a message. Shannon proved, *inter alia*, that entropy gives a precise estimate of how

much can be achieved by data compression. Combined with the famous Huffmann construction of the shortest code, this result of Shannon leads to a simple technique of data compression, presented in Chapters 2 and 3. (However, data compression is restricted to the case of information sources without memory.) The fourth chapter discusses the Fundamental Theorem of Shannon, which states that for every channel there exist error-correcting codes which remove noise while keeping the redundancy within the channel capacity. This result is purely theoretical: no algorithm for finding such codes has ever been found. The theory of error-correcting codes today has a lesser goal, viz., constructing codes with a reasonable redundancy and a fast decoder.

Constructions of efficient error-correcting and error-detecting codes with fast decoders are presented in the second part of the book. Some of the constructions require a deeper background in modern algebra and geometry, and we provide a thorough presentation of the relevant topics. The most important classes of error-correcting codes are the following:

Hamming codes (Chapter 5), perfect codes for single errors;

Reed-Muller codes (Chapter 9), multiple-error-correcting codes with a particularly efficient and easily implemented decoder;

Golay code (Chapter 10), the unique perfect code for triple errors;

BCH codes (Chapters 12 and 13), strong multiple-error-correcting codes with a fast decoder;

Convolutional codes (Chapter 14), multiple-error-correcting codes with memory.

The last part of the book is a short introduction to modern cryptography, stressing the role which error-correcting codes play here. Some of the well-known secret codes used in cryptography are based on constructions of error-correcting codes (e.g. the cryptosystem of McEliece, see 15.3). However, the main relation between cryptography and error-correcting codes is that, since noise is fatal for decryption, secret codes are usually combined with error-correcting codes. Furthermore, since encryption is costly, secret codes are usually combined with data compression.

The book is organized in chapters numbered consecutively throughout the three parts. Each chapter is divided into sections, and cross-references are always related to the number of section. For example, Theorem 3.2 means (the only) theorem in Section 3.2 of Chapter 3.

Part I

Coding and Information Theory

This page intentionally left blank

Chapter 1

Coding and Decoding

We are often faced with the task of converting a message, i.e., a sequence of symbols from a finite set (called a *source alphabet*), into a binary message, i.e., a sequence of 0's and 1's. The most common method is to translate each source symbol into a binary word. [*Word* means precisely a finite sequence; we often write $a_1 a_2 \ldots a_n$ instead of the more precise $(a_1, a_2, \ldots, a_n)$. Thus, 00101 is an example of a binary word.] Then the message is encoded symbol by symbol: we simply concatenate the words corresponding to the first, second, etc., symbol of the source message. The question of *how* to encode the source symbols is very important. There are two major criteria: we want to compress data, i.e., we want the resulting binary message to be as concise as possible, and we want to protect information against noise. These two requirements are rather contradictory, since by compressing data in the presence of noise, we are apt to increase, rather than decrease, the loss of information. In the present part, we therefore disregard noise: assuming that no errors occur, we try to find a concise code.

In the present chapter we introduce the important class of instantaneous codes, i.e., codes which can be decoded letter by letter, and we show how to construct such codes. The construction of the shortest code will be presented in Chapter 2.

1.1 Coding

Definition. *Given finite sets A* (source alphabet) *and B* (code alphabet), *a* coding *is a rule assigning to each source symbol exactly one word in the code alphabet, i.e., a function from A to the set of all words in B. We speak about* binary coding *if the code alphabet B has two symbols.*

Source Symbol	Code Word
1	11000
2	10100
3	01100
4	10010
5	01010
6	00110
7	10001
8	01001
9	00101
0	00011
Decoding	01247

Figure 1: 2-out-of-5 code

Example of Binary Coding. Observe that among binary words of length 5, the number of those having two 1's is $\binom{5}{2} = 10$. This can be used to the *2-out-of-5 code* of decimal digits, see Figure 1.

The message "173" has the following code: 110001000101100. Observe that no space is left between the code words since "space" is a code symbol too. Thus, for example, the famous Morse code has code alphabet $B = \{ \cdot , — , \text{space} \}$.

How do we decode the 2-out-of-5 code? Of course, the first five binary digits correspond to the first decimal one, and after decoding them, we proceed to the second group of five binary digits, etc. A helpful mnemonic rule: use 01247 as a "weight" of the five columns, and add the weights of all 1's in your words. Examples: $11000 \longmapsto 0+1 = 1$ and $01100 \longmapsto 1+2 = 3$. Unfortunately, 0 is an exception.

Remark. Given a coding (i.e., a function K from $A = \{a_1, \ldots, a_n\}$ to the set of all words in B), the words $K(a_1), \ldots, K(a_n)$ are called *code words*, and the set of all code words is called a *code*. When the concrete symbol in A is not important, "code" and "coding" are usually identified.

1.2 Unique Decoding

Definition. *For each coding K (of source symbols), we define the* coding of source messages *as the rule K^*, which to each word $x_1 x_2 \ldots x_m$ in the*

source alphabet assigns the word $K^*(x_1x_2\ldots x_m) = K(x_1)K(x_2)\ldots K(x_m)$ *obtained by concatenation of the code words* $K(x_i)$, $i = 1, \ldots, m$.

The coding K *is said to be* uniquely decodable *provided that arbitrary two distinct source messages have distinct codes. In other words, provided that* K^* *is one-to-one.*

For example, the 2-out-of-5 code is uniquely decodable. The assignment of a binary word to "173" is a sample of coding source messages.

In contrast, the following coding

$$a \longmapsto 00 \quad b \longmapsto 10 \quad c \longmapsto 101 \quad d \longmapsto 110 \quad e \longmapsto 1001$$

is not uniquely decodable: try to decode 10110.

We now introduce two important types of uniquely decodable codes.

1.3 Block Codes and Instantaneous Codes

We now introduce two important types of codes: instantaneous codes, which are codes of variable word lengths decodable symbol per symbol, and block codes, which are the special case of instantaneous codes with constant word length:

Definition. **(1)** *A coding using only pairwise distinct code words of a certain length* n *is called a* block coding *of length* n.

(2) *A coding is called* instantaneous *provided that no code word is a prefix of another code word; i.e., if a source symbol has a code* $b_1b_2\ldots b_n$ *then no other source symbol has a code* $b_1b_2\ldots b_nb_{n+1}\ldots b_m$.

Remark. Block codes (e.g., the 2-out-of-5 code above) are very convenient for decoding since we know in advance which code symbols correspond to the first (second, third, etc.) source symbol. And they are certainly efficient whenever all source symbols appear with equal frequency. However, if the frequencies of various source symbols differ substantially, then block codes become clumsy, and it is preferable to use instantaneous codes of variable lengths of words.

Examples

(1) The famous *Morse code* is exhibited in Figure 2. This is an instantaneous code with the code alphabet { · , — , space }. Since "space" is only used at the end of each code word, the decoding procedure is simple: we always look for the first "space". There is an obvious reason for not using a block code: the frequency of, say, "E" in the English language is much higher than that of "F".

A	·—	N	—·
B	—···	O	———
C	—·—·	P	·——·
D	—··	Q	——·—
E	·	R	·—·
F	··—·	S	···
G	——·	T	—
H	····	U	··—
I	··	V	···—
J	·———	W	·——
K	—·—	X	—··—
L	·—··	Y	—·——
M	——	Z	——··

Figure 2: Morse code

(2) An important example of a block code is the *octal code*:

0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

(3) Suppose that we are to find a binary coding for the alphabet $\{0, 1, 2, 3\}$, and we observe that 0 appears much more often in source messages than any other symbols. Then the following coding seems reasonable:

$$0 \longmapsto 0 \quad 1 \longmapsto 01 \quad 2 \longmapsto 011 \quad 3 \longmapsto 111.$$

We can decode quite easily: count the number of 1's among the last three symbols of the encoded message. If the number is i, then the last source symbol is i. However, the above coding is not instantaneous. Indeed, when receiving a long message

$$011111111111\ldots,$$

we will not know whether the first source symbol is 0, 1, or 2 until the message stops.

1.4 Some Important Block Codes

Long binary codes are difficult to handle. It is thus often suitable to group the binary symbols: we get shorter codes in more complex alphabets. For example, by forming groups of three symbols, we obtain the octal code, see 1.3. Representation by the octal code is usually indicated by the subscript 8. Example:

$$(01)_8 = 000001.$$

By forming groups of four binary symbols, we obtain the *hexadecimal code* in Figure 3.

Binary	Hexadecimal	Binary	Hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Figure 3: Hexadecimal code

A very important code used for a standard binary representation of alphabetic and numeric symbols is the ASCII code* (Figure 4). It has $2^7 = 128$ source symbols encoded into binary words of length 8: the first seven symbols carry the information, and the eighth one is set in such a way that the *parity* is even (i.e., each code word contains an even number of 1's). The role of this eighth symbol is to enable detection of single errors—we explain this in detail in Chapter 5. For example, the letter A has code

$$\text{A} \longleftrightarrow \underbrace{1000001}_{\substack{\text{inform.}\\ \text{symbols}}}\underbrace{0}_{\substack{\text{check}\\ \text{symbol}}},$$

which in Figure 4 is represented by its seven information bits: $1(01)_8 = 1\,000\,001$.

*American Standard Code for Information Interchange

Source Symbol	Code	Source Symbol	Code	Source Symbol	Code	Source Symbol	Code
@	$1(00)_8$	‘	$1(40)_8$	NUL	$0(00)_8$	SP	$0(40)_8$
A	$1(01)_8$	a	$1(41)_8$	SOH	$0(01)_8$	!	$0(41)_8$
B	$1(02)_8$	b	$1(42)_8$	STX	$0(02)_8$	"	$0(42)_8$
C	$1(03)_8$	c	$1(43)_8$	ETX	$0(03)_8$	#	$0(43)_8$
D	$1(04)_8$	d	$1(44)_8$	EOT	$0(04)_8$	$	$0(44)_8$
E	$1(05)_8$	e	$1(45)_8$	ENQ	$0(05)_8$	%	$0(45)_8$
F	$1(06)_8$	f	$1(46)_8$	ACK	$0(06)_8$	&	$0(46)_8$
G	$1(07)_8$	g	$1(47)_8$	BEL	$0(07)_8$	’	$0(47)_8$
H	$1(10)_8$	h	$1(50)_8$	BS	$0(10)_8$	(	$0(50)_8$
I	$1(11)_8$	i	$1(51)_8$	HT	$0(11)_8$	)	$0(51)_8$
J	$1(12)_8$	j	$1(52)_8$	LF	$0(12)_8$	*	$0(52)_8$
K	$1(13)_8$	k	$1(53)_8$	VT	$0(13)_8$	+	$0(53)_8$
L	$1(14)_8$	l	$1(54)_8$	FF	$0(14)_8$	,	$0(54)_8$
M	$1(15)_8$	m	$1(55)_8$	CR	$0(15)_8$	−	$0(55)_8$
N	$1(16)_8$	n	$1(56)_8$	SO	$0(16)_8$	.	$0(56)_8$
O	$1(17)_8$	o	$1(57)_8$	SI	$0(17)_8$	/	$0(57)_8$
P	$1(20)_8$	p	$1(60)_8$	DLE	$0(20)_8$	0	$0(60)_8$
Q	$1(21)_8$	q	$1(61)_8$	DC1	$0(21)_8$	1	$0(61)_8$
R	$1(22)_8$	r	$1(62)_8$	DC2	$0(22)_8$	2	$0(62)_8$
S	$1(23)_8$	s	$1(63)_8$	DC3	$0(23)_8$	3	$0(63)_8$
T	$1(24)_8$	t	$1(64)_8$	DC4	$0(24)_8$	4	$0(64)_8$
U	$1(25)_8$	u	$1(65)_8$	NAK	$0(25)_8$	5	$0(65)_8$
V	$1(26)_8$	v	$1(66)_8$	SYN	$0(26)_8$	6	$0(66)_8$
W	$1(27)_8$	w	$1(67)_8$	ETB	$0(27)_8$	7	$0(67)_8$
X	$1(30)_8$	x	$1(70)_8$	CAN	$0(30)_8$	8	$0(70)_8$
Y	$1(31)_8$	y	$1(71)_8$	EM	$0(31)_8$	9	$0(71)_8$
Z	$1(32)_8$	z	$1(72)_8$	SUB	$0(32)_8$	:	$0(72)_8$
[	$1(33)_8$	{	$1(73)_8$	ESC	$0(33)_8$	;	$0(73)_8$
\	$1(34)_8$	|	$1(74)_8$	FS	$0(34)_8$	<	$0(74)_8$
]	$1(35)_8$	}	$1(75)_8$	GS	$0(35)_8$	=	$0(75)_8$
^	$1(36)_8$	~	$1(76)_8$	RS	$0(36)_8$	>	$0(76)_8$
_	$1(37)_8$	DEL	$1(77)_8$	US	$0(37)_8$	?	$0(77)_8$

Figure 4: ASCII code (7 information bits)

Let us finally mention an "everyday" code we meet in most textbooks. It is called the *international standard book number*, ISBN, and it is a block code of length 10. (Various hyphens are often inserted between the symbols, but we can ignore them here since they are used just for optical orientation.) The code alphabet has 11 symbols: 0, 1, ..., 9 and X (read: ten). For example, the book of Lin and Costello (1983) has

ISBN 0-13-283796-X.

The first number 0 denotes the country (USA), 13 denotes the publisher (Prentice-Hall), and the next six digits are assigned by the publisher as an identification number of the book. The last symbol is a check symbol (analogously as in the ASCII code above). It is set in such a way that for each ISBN code word $a_1a_2a_3\ldots a_9a_{10}$, the sum

$$\sum_{i=1}^{10} ia_{11-i} = 10a_1 + 9a_2 + 8a_3 + \cdots + 2a_9 + a_{10}$$

be divisible by 11. For example, in the ISBN above:

$$10\times 0 + 9\times 1 + 8\times 3 + 7\times 2 + 6\times 8 + 5\times 3 + 4\times 7 + 3\times 9 + 2\times 6 + 1\times 10 = 132 = 11\times 12.$$

Some publishers have a three-digit identification (e.g., Wiley-Interscience has 471) and then they assign a five-digit number to each publication.

1.5 Construction of Instantaneous Codes

Suppose you want to construct a binary instantaneous code of the source alphabet $\{a_1,\ldots,a_n\}$. It is sufficient to specify the lengths $d_1,\ldots,d_n$ of the expected code words. In fact, we can certainly assume that $d_1 \le d_2 \le \cdots \le d_n$. Then we choose an arbitrary binary word $K(a_1)$ of length d_1. Next we choose a binary word $K(a_2)$ of length d_2, but we avoid all those which have the prefix $K(a_1)$. This is possible: the number of all binary words of length d_2 is 2^{d_2}. The number of those having the prefix $K(a_1)$ is $2^{d_2-d_1}$ (because you can choose the d_2-d_1 digits remaining after the prefix $K(a_1)$ arbitrarily). Since $2^{d_2} \ge 2^{d_2-d_1}+1$, we have at least one choice of $K(a_2)$.

Next, we want to choose a word of length d_3 which has neither prefix $K(a_1)$ nor $K(a_2)$. Thus, from the 2^{d_3} possible words, we must avoid all the $2^{d_3-d_1}$ words with the prefix $K(a_1)$ and all the $2^{d_3-d_2}$ words with the prefix $K(a_2)$. This is possible if (and only if)

$$2^{d_3} \ge 2^{d_3-d_1} + 2^{d_3-d_2} + 1.$$

Dividing the last inequality by 2^{d_3}, we obtain

$$1 \geq 2^{-d_1} + 2^{-d_2} + 2^{-d_3}.$$

Analogously, we can see that the following inequality

$$1 \geq 2^{-d_1} + 2^{-d_2} + \cdots + 2^{-d_n}$$

makes it possible to fulfil our task. It turns out that this inequality is both necessary and sufficient for a construction of instantaneous codes.

Example. We want to find an instantaneous code with the same lengths of code words as that in Example 1.3(3) above. This is possible since

$$1 \geq 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3}.$$

Here is such a code:

$$0 \longmapsto 0 \quad 1 \longmapsto 10 \quad 2 \longmapsto 110 \quad 3 \longmapsto 111.$$

1.6 Kraft's Inequality

Theorem. *Given a source alphabet of n symbols and a code alphabet of k symbols, then an instantaneous code with given lengths $d_1, d_2, \ldots, d_n$ of code words exists, whenever the following* Kraft's inequality

$$k^{-d_1} + k^{-d_2} + \cdots + k^{-d_n} \leq 1$$

is fulfilled.

PROOF. We can assume that the source alphabet $\{a_1, a_2, \ldots, a_n\}$ is presented in the order imposed by the lengths of the expected code words; i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$. We define instantaneous coding K by the following induction:

(1) Choose an arbitrary word $K(a_1)$ of length d_1.

(2) Suppose $K(a_1)$, $K(a_2)$, $\ldots$, $K(a_{s-1})$ have been chosen. Then choose an arbitrary word $K(a_s)$ of length d_s with no prefix among $K(a_1)$, $\ldots$, $K(a_{s-1})$. This is possible: the number of words with the prefix $K(a_i)$ is $k^{d_s - d_i}$, and thus we have a choice of

$$k^{d_s} - \sum_{i=1}^{s-1} k^{d_s - d_i}$$

words. From Kraft's inequality, we get

$$1 - \sum_{i=1}^{s-1} k^{-d_i} \geq k^{-d_s}$$

and multiplying by k^{d_s}, we conclude

$$k^{d_s} - \sum_{i=1}^{s-1} k^{d_s - d_i} \geq 1.$$

□

1.7 McMillan's Theorem

McMillan's Theorem. *Every uniquely decodable coding satisfies Kraft's inequality.*

Remark. We see that Kraft's inequality is not only sufficient, but also necessary for the construction of an instantaneous code. However, McMillan's Theorem says much more: instantaneous codes are just as efficient as uniquely decodable codes. More precisely, for every uniquely decodable code there exists an instantaneous code with the same lengths of code words.

PROOF. Let K be a uniquely decodable coding. Denote by d_i the length of the code word $K(a_i)$, $i = 1, 2, \ldots, n$. Observe that for each number $j = 1, 2, 3, \ldots$, we can form exactly k^j words of length j in the (k-symbol) code alphabet. By unique decodability, the number of source messages $a_{i_1} a_{i_2} \ldots a_{i_r}$ whose code has length j cannot exceed k^j. The length of the code is $d_{i_1} + d_{i_2} + \cdots + d_{i_r}$. Thus, we observe that the number of all sums of the form

$$d_{i_1} + d_{i_2} + \cdots + d_{i_r} = j \tag{1.7.1}$$

is smaller or equal to k^j.

It is our task to prove that the number

$$c = \sum_{i=1}^{n} k^{-d_i}$$

is smaller or equal to 1. For this, we will verify that the numbers $\frac{c^r}{r}$ are bounded for all $r = 1, 2, 3, \ldots$. In fact, each number $c > 1$ clearly fulfils

$\lim_{r\to\infty} \frac{c^r}{r} = \infty$, and, therefore, the theorem will then be proved. Let us compute the powers of c:

$$c^2 = \left(\sum_{i=1}^{n} k^{-d_i}\right)\left(\sum_{j=1}^{n} k^{-d_j}\right) = \sum_{i,j=1}^{n} k^{-(d_i+d_j)}$$

and, in general,

$$c^r = \sum_{i_1,\ldots,i_r=1}^{n} k^{-(d_{i_1}+d_{i_2}+\cdots+d_{i_r})}. \tag{1.7.2}$$

We can reorder the last sum by collecting all the summands k^{-j}, where j satisfies (1.7.1). The largest possible j is $j = d + d + \cdots + d = rd$, where $d = \max(d_1, \ldots, d_n)$. As observed above, the number of all summands k^{-j} in sum (1.7.2) is smaller or equal to k^j. Thus,

$$c^r \leq \sum_{j=1}^{rd} k^j \cdot k^{-j} = \sum_{j=1}^{rd} 1 = rd.$$

Consequently, $\frac{c^r}{r} \leq d$, which proves that $c \leq 1$. □

Exercises

1A What is the smallest length of a block code with the same source alphabet $\{A, B, \ldots, Z\}$ and the same code alphabet $\{\cdot, —, \text{space}\}$ as the Morse code?

1	...	01
2	...	011
3	...	10
4	...	1000
5	...	1100
6	...	0111

Figure 5

A	...	1010
B	...	001
C	...	101
D	...	0001
E	...	1101
F	...	1011

Figure 6

1B Is the code in Figure 5 uniquely decodable? Is it instantaneous? Can you find an instantaneous code with the same lengths of code words?

1C Is the code in Figure 6 uniquely decodable? If not, exhibit two source messages with the same code.

1D Is the code in Figure 7 uniquely decodable?

0	...	AA
1	...	AABAB
2	...	ABBBBB
3	...	ABABA
4	...	ABBAA
5	...	BABBA
6	...	BBBAB
7	...	AAAABB
8	...	AAAABA
9	...	AAAAAB

Figure 7

A	...	001	A	...	00
B	...	1001	B	...	10
C	...	0010	C	...	011
D	...	1110	D	...	101
E	...	1010	E	...	111
F	...	01110	F	...	110
G	...	0101	G	...	010

Figure 8

1E Can you decide unique decodability of the two codes in Figure 8 by using Kraft's inequality?

1F Construct a binary instantaneous code for the following source alphabet with the prescribed lengths of code words:

Symbol	A	B	C	D	E	F	G	H	I	J	K	L
Length	2	4	7	7	3	4	7	7	3	4	7	7

1G Construct a ternary (three code symbols) instantaneous code for the following source alphabet with the prescribed lengths of code words:

Symbol	1	2	3	4	5	6	7	8	9	0
Length	1	3	3	3	3	2	2	2	2	2

1H How many code symbols are needed if the following source alphabet is to be encoded into an instantaneous code with the prescribed lengths of code words:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	2	2	2	1	2	2	2	1	2	2	2	2	2	1	2

1I Prove that for each instantaneous code for which Kraft's inequality is not an equality, it is possible to add a new source symbol and extend the given code to an instantaneous code (with the same code alphabet). Demonstrate this on the code found in Exercise 1.6.

Notes

Coding theory and information theory have their origin in the fundamental work of Claude E. Shannon (1948), reprinted in Slepian (1974), although many of the ideas were understood and used before. The small interesting book of Richard N. Hamming (1980), one of the founders of coding theory, provides further historical remarks.

Kraft's inequality was formulated by Kraft (1949). The source of McMillan's Theorem is McMillan (1956), the present proof is from Karush (1961).

Chapter 2

Huffman Codes

We have mentioned that if the frequencies of source symbols vary, then instantaneous codes can be preferable to block codes: the most frequent symbols will be encoded into the shortest code words. We now make these considerations more precise. If the frequencies of source symbols are known exactly (i.e., if the probability distribution of source symbols in messages has been determined), we want to find the most efficient coding.

2.1 Information Source

Definition. *An* information source *is a source alphabet together with a probability distribution; i.e., a set* $\{a_1, \ldots, a_n\}$ *together with numbers* $P(a_1), \ldots, P(a_n)$ *satisfying* $\sum_{i=1}^{n} P(a_i) = 1$ *and* $0 \leq P(a_i) \leq 1$.

More precisely, we should speak about a discrete, zero-memory information source: discrete because we have discrete source symbols, and zero-memory because we assume that the source symbols appear independently. In other words, the probabilities $P(a_i)$ fully describe the statistics of the source messages, and the probability of a message $a_{i_1} a_{i_2} \ldots a_{i_r}$ can be computed from the probabilities of the individual symbols:

$$P(a_{i_1} a_{i_2} \ldots a_{i_r}) = P(a_{i_1})\, P(a_{i_2}) \ldots P(a_{i_r}).$$

2.2 Huffman Codes

Let K be a coding of an information source. That is, for each source symbol a_i, we have a code word $K(a_i)$ and we know the probability $P(a_i)$