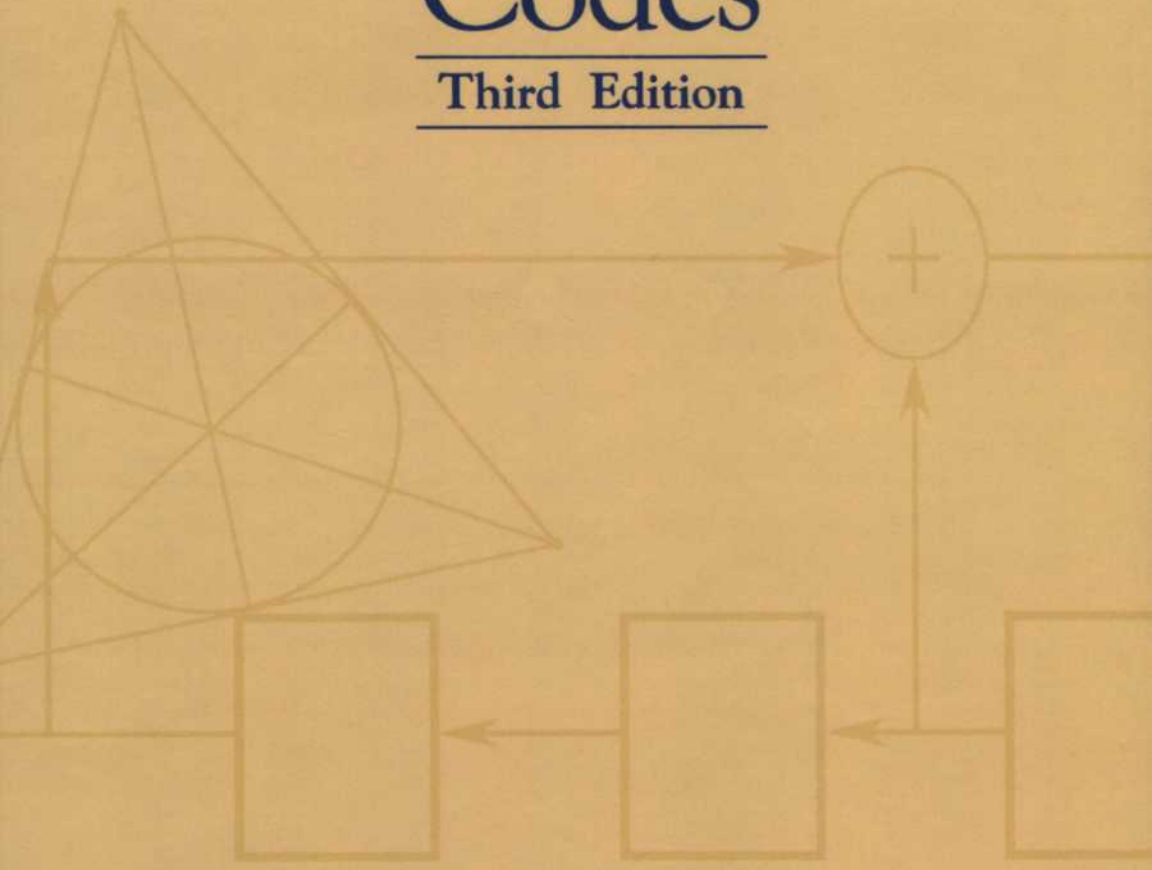
Introduction to the Theory of Error-Correcting Codes

Third Edition



Vera Pless

Wiley-Interscience Series in Discrete Mathematics and Optimization

This page intentionally left blank

Introduction to the Theory of Error-Correcting Codes

WILEY-INTERSCIENCE
SERIES IN DISCRETE MATHEMATICS AND OPTIMIZATION

A complete list of titles in this series appears at the end of this volume.

Introduction to the Theory of Error-Correcting Codes

THIRD EDITION

VERA PLESS
University of Illinois at Chicago

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

This book is printed on acid-free paper. ♾

Published simultaneously in Canada.

Library of Congress Cataloging in Publication Data:

Pless, Vera.
Introduction to the theory of error-correcting codes / Vera Pless.
—3rd ed.
p. cm.— (Wiley-Interscience series in discrete mathematics and optimization)
"A Wiley-Interscience publication."
Includes bibliographical references (p. -) and index.
ISBN 0-471-19047-0 (alk. paper)
1. Error-correcting codes (Information theory) I. Title.
II. Series.
QA268.P55 1998
003'.54—dc21 97-49834

To my children
Nomi, Ben, and Dan
and grandchildren
Lilah, Evie, and Becky

This page intentionally left blank

Contents

This page intentionally left blank

Preface

This book arose out of a two-quarter sequence in error-correcting codes that I taught at the University of Illinois Circle Campus. It is intended for undergraduates and graduate students in mathematics, computer science, or electrical engineering. The only requirement is an elementary course in linear algebra. An appendix, which covers some of the linear algebra needed, is provided to supplement such a course. A modern algebra course is not necessary but would probably be helpful. If the algebra course is taken concurrently with the coding course, the latter could provide motivation and many concrete examples. Instructors can determine the pace at which to proceed by the mathematical backgrounds of their students.

The theory of error-correcting codes started as a subject in electrical engineering with Shannon's classic papers in 1948. It has since become a fascinating mathematical topic, and part of the fascination has been the use of many varied mathematical tools to solve the practical problems in coding. This book attempts to demonstrate this process. Understanding how one might go about finding mathematical techniques to solve applied problems is useful to students who might sometime encounter such problems. Because the subject is relatively new, there are many open problems in coding. Some of these are mentioned in this book. Whenever possible, the most elementary proofs or approaches are used.

Since the first edition was written, practical uses of error-correcting codes have proliferated. In addition to many uses in communication systems, error-correcting codes are widely used in modern memory devices, have many uses in computer systems, and also provide the high fidelity on many compact disc players. Although the technology is changing rapidly, the fundamental principles of coding remain the same.

This book is about linear block codes, and general background material is given in the first two chapters, including an introduction to such specific linear codes as Hamming codes, Reed-Muller codes, and Golay codes. Chapter 3 raises the problem of how to correct double errors, which leads to the necessity of using finite fields, the topic of Chapter 4. Chapter 5 covers the important class of cyclic codes. Chapter 6 talks about an interesting family of cyclic codes, quadratic residue codes, and also about the group of a

code. Chapter 7 discusses the practical BCH codes. Chapter 8 is on weight distributions, particularly of self-dual codes, which leads to designs in codes in the next chapter. Chapter 9 concludes with a recent connection between codes and combinatorial games. Chapter 10 includes a proof of the uniqueness of the Golay code, how to glue codes together, and the new way to handle nonlinear codes using Z_4 codes. This is more than can be covered in a two-quarter or one-semester course on coding. Instructors should choose what they like after covering the first five chapters and much of chapters six and seven if that is all the time available, or else spend a whole year on coding.

The books about and related to coding which my students have consulted are E. R. Berlekamp, *Algebraic Coding Theory* [30]; P. J. Cameron and J. H. van Lint, *Designs, Graphs, Codes and Their Links* [46]; J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups* [33]; R. Hill, *A First Course in Coding Theory* [34]; J. H. van Lint, *Introduction to Coding Theory* [40]; and of course, F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes* [15].

I want to thank all the faculty and students who used the text and offered their comments. I have kept track of these and tried to incorporate them. Since I have taught this course myself, there were items I wanted to change, and I also found better ways of presenting topics. In this revision many mistakes are corrected, several new topics introduced, and many new problems added throughout. I want to thank Noburn Ito and Dan Pritkin, in particular, for their detailed and valuable suggestions.

Introduction to the Theory of Error-Correcting Codes

This page intentionally left blank

1

Introductory Concepts

1.1 INTRODUCTION

The subject of error-correcting codes arose originally in response to practical problems in the reliable communication of digitally encoded information. Claude Shannon's paper "A Mathematical Theory of Communication" [28], written in 1948, started the discipline in electrical engineering called information theory, and also the branch of it called error-correcting codes. Since then algebraic coding has developed many connections with portions of algebra and combinatorics. Sophisticated mathematical techniques have proved useful for coding and coding problems, and the results have interested mathematicians. Now algebraic coding is also a mathematical topic with the added feature that its recent practical origins have provided motivation for many of its main concerns.

We here think of a message as a block of symbols from a finite alphabet. A commonly used alphabet is the set of two symbols 0 and 1, and so we start with that. A possible message is 1001. This can represent a number such as 759, a letter such as A, or a complete message such as "The yellow cat is sick." This message is transmitted over a communications channel that is subject to some amount of noise. The object of an error-correcting code is to add redundancy to the message in an analytic fashion so that the original message can be recovered if it has been garbled. This is commonly done in ordinary speech when people repeat things in many different ways in order to be properly understood. Consider now the diagram of the communications channel (Figure 1.1).

The first box contains the message, in our case 1001, which we say represents "The spacecraft is approaching from the north." This message then enters the encoder where the redundancy digits 101 are added so that the message can be corrected if it becomes distorted when communicated. The message is transmitted over the channel, where it is subject to noise. When noise hits the message, a 0 is changed to a 1 or a 1 to a 0. In our message the first digit was changed by noise. Then the receiver is either completely confused—possibly 0001101 does not stand for any message—or else she is misinformed—0001101 could represent the message "The spacecraft is approaching from the east." Now the received message enters the

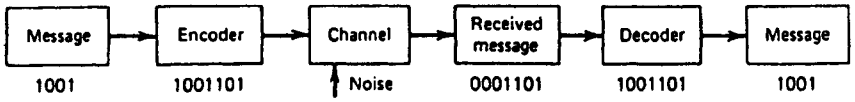


Figure 1.1. Communications channel.

decoder where, due to the redundancy added, the original message can be recovered. Error-correcting coding is the art of adding redundancy efficiently so that most messages, if distorted, can be correctly decoded.

We can think of the communications channel as a real communications channel or as data stored in a computer that deteriorate with time. Furthermore, although we may think the channel's reliability is quite good, our need for reliable communications is great. For example, in communicating with a satellite any mistake could be disastrous.

Communications channels where error-correcting codes are used are too numerous to mention. Codes are used on telephone lines and computer links. Black and white pictures were transmitted from several Mariner space probes using error-correcting codes, as were color pictures from recent Voyager journeys. Error-correcting codes give the high fidelity on compact discs. Errors arise from a variety of causes, some of which are human, equipment failure, lightning, interference, or scratches on discs. Error-correcting codes are also used for data compression. Indeed, their uses are ever expanding.

We make a distinction between detecting and correcting errors. It is much easier to detect errors than to correct them, and at times even detection is useful. For example, if there is a feedback channel and enough time, we can ask for the message to be sent again. This is not always possible, such as with data stored on magnetic tape. In many real-time communications, it is often necessary to correct errors, and that is our emphasis here.

One of the simplest channels is the binary symmetric channel BSC. It has no memory, and it receives and transmits two symbols, 0 and 1. The BSC has the property that with probability q a transmitted digit will be received correctly, and with the probability $p = 1 - q$ it will not be. This can be illustrated as shown in Figure 1.2. We call p the *symbol error probability.*

In order to see the problems we face, we now attempt to construct some simple codes. Suppose that our message is 1001. If we add no redundancy and transmit this message and an error occurs, there is no way to detect it. If

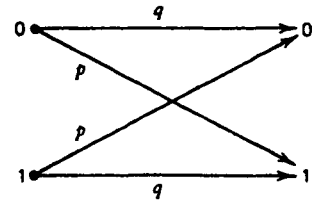


Figure 1.2. Binary symmetric channel.

we start with the modest aim of detecting single errors, we can begin by repeating our message once; 10011001. If an error occurs and we receive, say, 10010001, then by comparing the two parts we can detect it. Still we do not know whether the error is in the first or second rendition. Also we are transmitting twice as many digits as we need, and if we merely wish to detect single errors, we can do better than this by using the *overall parity check*. A parity check is accomplished by adding an extra digit to the message. It is a 0 if there are an even number of 1's, and a 1 if there are an odd number of 1's. Then we transmit the message 10010, and if we receive, say 10000, we can detect that an error has occurred since we see an odd number of 1's. The overall parity check is widely used in computers and elsewhere.

In error correcting our first aim is to correct single errors since these are the most probable. We will then correct double errors, and then triple errors, and so on, correcting as many as we can. If we now make a naive attempt to correct single errors, we could transmit our message three times; 100110011001. Then if an error occurs and we receive 100100011001, we can take a majority count of the disagreeing digits 1, 0, 1, and decide that most likely a 1 was sent. However, we have now transmitted three times as much as we need, although we can correct some double and triple errors (which ones?). A significant improvement occurs with the Hamming [7, 4] code. We describe this code by the following four codewords.

1	2	3	4	5	6	7
1	0	0	0	0	1	1
0	1	0	0	1	0	1
0	0	1	0	1	1	0
0	0	0	1	1	1	1

We think of the first four positions as the information positions and the last three as the redundancy positions. The first codeword represents the message 1000, the second represents the message 0100, and so on. We write each codeword either as a block of seven 0's and 1's, say 1000011, or in the more usual vector form as $(1, 0, 0, 0, 0, 1, 1)$. We can obtain more codewords by adding these four vectors where in each addition we add corresponding components mod 2. So, for example, $(1, 0, 0, 0, 0, 1, 1) + (0, 1, 0, 0, 1, 0, 1) = (1, 1, 0, 0, 1, 1, 0)$. This codeword represents the message 1100. In this fashion we can encode all $2^4 = 16$ messages consisting of four symbols, each either 0 or 1. In order to decode this code, we consider the following three decoding sequences:

$$\mathbf{a} = 0001111,$$
$$\mathbf{b} = 0110011,$$
$$\mathbf{c} = 1010101,$$

and we use the inner product of two vectors $\mathbf{x} = (x_1, \ldots, x_7)$ and $\mathbf{y} = (y_1, \ldots, y_7)$ defined by $\sum_{i=1}^{7} x_i y_i \pmod 2$. If our message is 1011, we encode it

as $\mathbf{x} = 1011010$. Suppose that an error occurs, and we receive $\mathbf{y} = 1010010$. We form the three inner products $\mathbf{y} \cdot \mathbf{a} = 1$, $\mathbf{y} \cdot \mathbf{b} = 0$, and $\mathbf{y} \cdot \mathbf{c} = 0$. In this order the symbols represent the binary number $100 = 4$. From this we conclude that the fourth digit is in error. This procedure is known as Hamming decoding, and it corrects any single error. Note that we do not even transmit twice the number of information digits.

Suppose that we are transmitting on a binary channel and that q, the probability of correct transmission, is .90 so that p, the probability of an error, is .10. Then the probability that a four-digit message will be transmitted correctly without any coding is $q^4 = .6561$. If the Hamming code is used, then the probability of correctly decoding a received vector is q^7 (no errors) + $7pq^6$ (one error) = .4783 + .3720 = .8503, a considerable improvement.

The word error rate P_{err} for a particular decoding scheme is the probability of incorrectly decoding a received message. For the Hamming code we see that P_{err} is $1 - (q^7 + 7pq^6) = .1497$.

Hamming decoding is so easy that it can be done quickly without a computer. Yet many intelligent people have worked for years to devise decoding schemes for other codes or classes of codes that will operate efficiently on computers. One of the codes that has received such attention is the famous Golay [23, 12] code. Here each codeword has 23 digits of which 12 are the information digits and 11 are the redundancy digits. It is known that this code can correct one, two, or three errors, but its decoding is often done by a computer and people have worked on numerous decoding schemes. To see why such effort is needed, consider the Hamming code repeated three times. Break any string of 12 information digits into portions of 4 digits each, then encode each of these portions using the Hamming code. We can consider this as a code with 12 information digits and 9 redundancy digits, 2 fewer than the Golay code, and we can decode it using Hamming decoding for each set of 7 digits. Decoding is easily done by hand. This code can only correct single errors and some double or triple errors if they occur in separate Hamming blocks, whereas the Golay code can correct double or triple errors wherever they are distributed. Larger codes that can correct more errors are more useful because the errors can be distributed in more ways, but they are harder to decode.

We can see some of the reasons for this from Table 1.1. In the first code, the Hamming code, we have a possible 128 received vectors, and we must decide which of the 16 codewords was sent. This is much more difficult for the second code, the Golay code, which has 8,388,608 possible received messages, which are to be decoded into one of 4096 codewords. The last code, which we see later; is a quadratic residue code; it presents even more difficulty. Often, for practical purposes, codes are needed whose lengths are in the hundreds.

The *rate* of a code is defined as the ratio of the number of information digits to the length. Thus the codes above have rates 4/7 for the first, 12/23

Table 1.1 Some Binary Codes

1	2	3	4	5
7	4	1	16	128
23	12	3	4,096	8,388,608
47	24	5	16,777,216	140,737,488,355,238

Column 1—length of code, n.
Column 2—number of information symbols, k.
Column 3—number of errors code can correct.
Column 4—number of messages in code = 2^k.
Column 5—total number of messages = 2^n.

for the second, and 24/47 for the third; all these numbers are close to 1/2. A very important result in coding is the surprising theorem of Claude Shannon. We cannot state it precisely without defining more concepts, but roughly it says that if the code rate is less than a number called "channel capacity," which measures the amount of information that a channel can transmit, then it is possible to transmit information with an arbitrarily small probability of error by using long enough codes. However, Shannon proved this theorem by probabilistic methods, not constructive ones, and one of the outstanding problems in coding is constructing families of codes with known properties that are as good as the theorem predicts. A "good" code is a code that can intrinsically correct many errors. For practical purposes, however, it is also important to be able to decode efficiently.

Since Shannon's theorem, coding theorists have constructed many good codes, devised ingenious decoding algorithms, and developed the theory of codes. A number of these good codes will be described in later chapters but we will not be able to cover all the known ones. The construction (in the 1980s) of new, good codes using methods of algebraic geometry has created much excitement in the coding community. Unfortunately, their study requires an extensive knowledge of algebraic geometry. But we will describe recent constructions using the integers modulo 4.

In this book we are concerned primarily with linear or algebraic codes where the errors are randomly distributed. Shannon's theorem is about nonlinear codes, but an analogous theorem has been demonstrated for linear codes so there are good reasons for studying them. Much more is known about them, and this knowledge can be used for decoding and storage. There are special codes for correcting bursts of errors, but even there the more general case provides useful information. There are codes where the messages are not broken into blocks but form a continuous stream. These are called convolutional codes, and they often find practical applications. We do not study them here because their structure is apparently quite different from block codes and possibly not as well understood.

Another interesting topic is how the original information is assigned its block of digits. There might be reasons based, for example, on frequency of

occurrence for such assignments. This topic is called source encoding and is not pursued here. We do, however, study linear codes with elements from any finite field, also "linear" codes over Z_4.

1.2 BASIC DEFINITIONS

In the last section we saw a specific binary code, the Hamming code, described by its generator matrix. We now define a linear code and describe two common ways to give such a code, one by a generator matrix and the other by a parity check matrix. Since we are often concerned with binary codes, we sometimes state our definitions separately for that case.

In order to define a binary linear code, we consider the space V of all n-tuples of 0's and 1's with addition of vectors component wise mod 2. So, for example, $(1,0,0,0,1,1) + (0,1,0,1,0,1) = (1,1,0,1,1,0)$. An $[n,k]$ *linear, binary code* is the set of all linear combinations of k independent vectors in V. The word linear means that if two (or more) vectors are in the code, so is their sum. A *nonlinear code* is just a set of vectors. Since we are mainly concerned with linear codes, the word code means a linear code. We could also define an $[n,k]$ binary code C by saying C is a k-dimensional subspace of V.

In the general case we let F be $GF(q)$, the finite field with q elements. This is described thoroughly in Chapter 4. We include it here for completeness. Then it is known that q must be a power of a prime. If q is itself a prime p, say $q = 2$ or 3, then F can be thought of as the set of p elements $0, 1, \ldots, p-1$ with the arithmetic operations performed mod p. A binary code is a code over $GF(2)$. The reader who has not encountered finite fields should think of these cases when $GF(q)$ is mentioned. An $[n,k]$ *code over* $GF(q)$ is a k-dimensional subspace of F^n, the space of all n-tuples with components from $F = GF(q)$.

Clearly an $[n,k]$ binary code has 2^k vectors or codewords in it. Since a code is a vector subspace, it can be given by a basis. The matrix whose rows are the basis vectors is called a *generator matrix*. Just as a subspace has more than one basis, a code has more than one generator matrix.

Consider, for example, the [5,3] binary code C_1 whose generator matrix is G_1:

$$G_1 = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

All the codewords in C_1 can be gotten from the linear combinations of these three vectors. A codeword in C_1 has three information positions. Any three positions where the columns of G_1 are independent can be taken as information positions. The first three positions certainly can be, but there are others. C_1 has $8 = 2^3$ codewords.

If G is a generator matrix of an $[n, k]$ code C, then any set of k columns of G that are independent is called an *information set* of C. If a certain set of k columns of one generator matrix is independent, then that set of columns of any generator matrix is independent. Any $[n, k]$ code with $k \neq n$ and fewer than $n - k$ zero columns has more than one information set. It is a fact that the information sets constructed from any generator matrix G' of C will be the same as those constructed from G.

Another way to describe a code is by parity check equations. Since in C_1 the first three positions are information positions, we can express the redundancy positions in terms of them. We let $(a_1, a_2, a_3, a_4, a_5)$ be any vector in C_1, and suppose that we know the information positions a_1, a_2, and a_3. Then the redundancy positions can be computed in terms of these as follows:

$$a_4 = a_1 + a_3$$
$$a_5 = a_1 + a_2 + a_3.$$

Any codeword $(a_1, a_2, \ldots, a_5)$ in C_1 satisfies these equations. They are the parity check equations for C_1. We can use them to write down all vectors in C_1. So, for example, if $a_1 = a_2 = 1$ and $a_3 = 0$, then $a_4 = 1 + 0 = 1$, $a_5 = 1 + 1 + 0 = 0$, and $(1, 1, 0, 1, 0)$ is in C_1. This is the sum of rows 1 and 2 of G_1.

A set of equations that give the redundancy positions in terms of the information positions are called *parity check equations*. We can express all such equations in terms of the parity check matrix. In order to do this, we use the inner product of two vectors. This is the same inner product we used in Hamming decoding.

If $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{v} = (v_1, \ldots, v_n)$ are two vectors in V over $GF(p)$, for p a prime, then the *inner product* of $\mathbf{u}$ and $\mathbf{v}$ is $\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i \pmod{p}$. The inner product is linear in both variables, $(\alpha_1 u_1 + \alpha_2 u_2) \cdot (\beta_1 v_1 + \beta_2 v_2) = \alpha_1 \beta_1 u_1 \cdot v_1 + \alpha_1 \beta_2 u_1 \cdot v_2 + \alpha_2 \beta_1 u_2 \cdot v_1 + \alpha_2 \beta_2 u_2 \cdot v_2$. Clearly $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$.

If $\mathbf{u} \cdot \mathbf{v} = 0$, we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal* to each other. For binary vectors this means that they have an even number of 1's in common. Let

$$H_1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Note that a vector $(a_1, a_2, a_3, a_4, a_5)$ is orthogonal to the first row of H_1 if $a_1 + a_3 + a_4 = 0$, which is the same as our first parity check equation $a_4 = a_1 + a_3$. Similarly being orthogonal to the second row of H_1 is the same as satisfying the second parity check equation. Hence we can now say that C_1 is the set of all 5-tuples that are orthogonal to each row of the parity check matrix H_1. This is exactly the same as saying that the vectors in C_1 satisfy the parity check equations given above.