The Froehlich/Kent

ENCYCLOPEDIA OF TELECOMMUNICATONS

VOLUME 3

Editor-in-chief: Fritz E. Froehlich

Coeditor: Allen Kent

The Froehlich / Kent

ENCYCLOPEDIA OF TELECOMMUNICATIONS

VOLUME 3

The Froehlich/Kent
ENCYCLOPEDIA OF TELECOMMUNICATIONS

Editor-in-Chief

Fritz E. Froehlich, Ph.D.
Professor of Telecommunications
University of Pittsburgh
Pittsburgh, Pennsylvania

Co-Editor

Allen Kent
Distinguished Service Professor of Information Science
University of Pittsburgh
Pittsburgh, Pennsylvania

Administrative Editor

Carolyn M. Hall
Pittsburgh, Pennsylvania

VOLUME 3

CODES FOR THE PREVENTION OF ERRORS to COMMUNICATIONS FREQUENCY STANDARDS

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

First published 1992 by Marcel Dekker, Inc.

Published 2021 by CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

ISBN 13: 978-0-8247-2902-8 (hbk)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Library of Congress Cataloging-in-Publication Data

The Froehlich/Kent Encyclopedia of Telecommunications / editor-in-chief, Fritz E. Froehlich ; co-editor, Allen Kent.
p. cm.
Includes bibliographical references and indexes.
ISBN 0-8247-2902-1 (v. 1 : alk. paper)
1. Telecommunication—Encyclopedias. I. Froehlich, Fritz E., II. Kent, Allen.
TK5102.E646 1990 90-3966
384′.03—dc20 CIP

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 90-3966

DOI: 10.1201/9781003209867

CONTENTS OF VOLUME 3

CONTRIBUTORS TO VOLUME 3

Walter R. Beam, Ph.D. Independent Consultant, Alexandria, Virginia: *Command, Control, and Communications Systems*

Richard E. Blahut, Ph.D. IBM Fellow, IBM, Federal Sector Division, Owego, New York: *Codes for the Prevention of Errors*

Amalie J. Frank, Ph.D. Science Division, Widener University, Chester, Pennsylvania: *Coding of Facsimile Images*

Richard R. Goldberg District Manager, Bellcore, Red Bank, New Jersey: *Common Channel Signaling*

Daniel J. Hunt Director, Education Affairs, Northern Telecom, Research Triangle Park, North Carolina: *Communication in Education*

Monica Krueger, MST University of Pittsburgh, School of Library and Information Science, Pittsburgh, Pennsylvania: *Colpitts, Edwin H.*

Linda J. Laskowski Vice President and General Manager, Information Provider Market, U S West Communications, Denver, Colorado: *Communication Gateway Services for Videotex*

Nancy K. Metzler Manager, Videotex, Information Provider Market, U S West Communications, Denver, Colorado: *Communication Gateway Services for Videotex*

Bob Neveln, Ph.D. Associate Professor of Mathematics and Computer Science, Science Division, Widener University, Chester, Pennsylvania: *Comma-Free and Synchronizable Codes*

Victor L. Ransom Division Manager (Retired), Bell Communications Research, Inc., Red Bank, New Jersey: *Communication Aids for People with Special Needs*

Laura S. Redmann Member of Technical Staff, Bell Communications Research, Inc., Red Bank, New Jersey: *Communication Aids for People with Special Needs*

David C. Rife, Ph.D. Senior Principal Engineer, Hayes Microcomputer Products, Inc., Norcross, Georgia: *Communication with Intelligent Modems*

Carl V. Ripa Managing Director, Telesector Resources Group, White Plains, New York: *Communications and Information Network Service Assurance*

Samuel R. Stein, Ph.D. Ball Corp., Broomfield, Colorado: *Communications Frequency Standards*

Seymour Stein, Ph.D. SCPE, Inc., Newton, Massachusetts: *Communication over Fading Radio Channels*

Richard A. Thompson, Ph.D. Professor of Telecommunications, University of Pittsburgh, Pittsburgh, Pennsylvania: *Communication Terminals*

Heather S. Tooker Director, Videotex, Information Provider Market, U S West Communications, Denver, Colorado: *Communication Gateway Services for Videotex*

John R. Vig, Ph.D. U.S. Army Electronics Technology and Devices Laboratory, Fort Monmouth, New Jersey: *Communications Frequency Standards*

Keith D. Wallin Manager, Videotex, Information Provider Market, U S West Communications, Denver, Colorado: *Communication Gateway Services for Videotex*

W. Bernard Wargotz, Ph.D. Undersea Systems Laboratory, AT&T Bell Laboratories, Holmdel, New Jersey: *Communication Printed Wiring Interconnection Technology*

Taieb F. Znati, Ph.D. Computer Science Department (Telecommunications), University of Pittsburgh, Pittsburgh, Pennyslvania: *Communication Protocols for Computer Networks: Fundamentals*

The Froehlich / Kent

ENCYCLOPEDIA OF TELECOMMUNICATIONS

VOLUME 3

Codes for the Prevention of Errors

Introduction

A profusion and variety of communication systems now exist to carry a massive amount of digital data between terminals. Alongside these communication systems are very large numbers of magnetic and optical tape and disk storage systems. The received signals in all communication and recording systems are always contaminated by thermal noise and, in practice, are also contaminated by various kinds of defects, non-Gaussian noise, burst noise, interference, and crosstalk. The communication system (or storage system) must transmit its data with very high reliability in the presence of these channel impairments. Bit error rates of as low as 1 bit error in 10^{10} bits (or even lower) are routinely specified.

Primitive communication and magnetic-storage systems seek to obtain low error rates by the simple expedient of transmitting high power in comparison to the noise. This simplistic approach may be adequate if the required bit error rate is not too stringent or if the data rate is low. Such systems, however, buy performance with the least expendable resources: power and spectral bandwidth.

In contrast, modern communication and storage systems obtain high performance by the use of elaborate message structures with complex cross checks built into the waveform. In some systems, it is not possible to determine where a particular bit resides in the channel waveform; the entire message is modulated into the waveform as a unit and an individual bit appears in a diffuse but recoverable way. The advantage of the modern communication waveforms is that high data rates can be transmitted reliably while keeping the transmitted power and spectral bandwidth small. This advantage is offset by the need for sophisticated computations in the receiver (and in the transmitter) to recover the message. However, such computations are now regarded as affordable using modern electronic technology. For example, some current telephone-line data modems use microprocessors in the receiver with approximately 500 machine cycles of computation per received data bit. Clearly, with this amount of computation in the receiver, the waveforms may have a very sophisticated structure and each individual bit can, indeed, be deeply buried in the waveform.

The codes described in this chapter are called *codes for the prevention of error*. This is a positive term that implies the role that such codes have in modern systems. The original term, *error-correcting codes*, is also used but suffers from the fact that it has a negative connotation. It implies that the code is used only to correct an unforeseen deficiency in the communication system whereas, in fact, the code is an integral part of the system design. Furthermore, the way some codes are usually used, it may be hard to find that point in the system where the errors that the code is correcting occur. It is a better description to say that the errors are prevented from ever happening.

Other, more neutral, terms for the codes in this article are *error-control codes* and *data-transmission codes*. The term *channel codes* may also be used,

DOI: 10.1201/9781003209867-1

but this term includes other kinds of codes, such as the Morse code, that have other functions in the communication system.

Codes Based on Hamming Distance

Given two sequences of the same length of symbols from some fixed-symbol alphabet, perhaps the binary alphabet $\{0,1\}$, we discuss how different those two sequences are. The most suggestive way to measure the difference between the two sequences is to count the number of places in the sequence in which they differ. This is called the *Hamming distance* between the sequences. The reason for choosing the term "distance" is to appeal to geometric intuition when constructing codes.

For example, consider the two sequences of length 6 over the decimal alphabet given by 0,1,2,3,4,5 and 0,1,4,5,4,5. The Hamming distance between these sequences is 2, which we denote symbolically by

$$d(012345, 014545) = 2$$

or

$$d(\mathbf{v},\mathbf{v}') = 2$$

where the vectors $\mathbf{v}$ and $\mathbf{v}'$ denote the two sequences.

We may also have two infinitely long sequences over some symbol alphabet. Again, the Hamming distance is defined as the number of places in which the two sequences are different. This Hamming distance then will be infinite unless the sequences agree everywhere except on a finite segment.

Although user data may be treated on the bottom level as a sequence of bits, often it is treated as a sequence of bytes. Frequently, a byte consists of 8 bits, but sometimes a communication system may be designed around a byte of r bits for some value of r other than 8. This data structure within the communication system is transparent to the user because the datastream is reformatted at the input and output of the modem.

When sequences are described at the byte level, the Hamming distance between two sequences is the number of byte positions in which they differ. Two bytes are different if they are not the same. They may differ in any possible way involving one or more bit positions. Thus, we speak of byte errors rather than of bit errors.

A *datastream* is a sequence of user data symbols, either bits or bytes. A *codestream* is a sequence of channel symbols, either bits or bytes. The user perceives that the datastream is being sent through the channel, but the codestream is actually sent.

For constructing the code, additional structure is defined on the datastream by segmenting it into pieces called *datawords* or *data frames*. Likewise, the codestream is segmented into pieces called *codewords* or *code frames*.

The encoder maps the datastream into the codestream. Codes are of two types: block codes and tree codes; the distinction between them due to the memory in the encoders of the tree codes.

Elementary Block Codes

A block code breaks the datastream into datawords, each consisting of k data symbols. The encoder maps each dataword into a codeword consisting of n code symbols. In brief, this is called an (n,k) block code. The ratio k/n is called the *code rate* and is denoted by R. Each block is encoded independently without interaction with earlier or later blocks. The successive codewords are simply concatenated to form the codestream for passage through the channel. For example, the compact disk encodes a dataword consisting of 28 8-bit bytes into a codeword consisting of 32 8-bit bytes. This is a (32,28) code on the alphabet of 8-bit bytes and has a rate $R = 0.875$.

There are 2^k codewords in a binary block code and q^k codewords in a block code over a q-ary alphabet. For the alphabet of 8-bit bytes, $q = 256$, so there are 256^k codewords in such a code. For the (32,28) code used on the compact disk, there are 256^{28} codewords in the code. A block code is designed such that the codewords are very different from each other so as to make the code resistant to channel errors. This dissimilarity is measured by the smallest Hamming distance between the two most similar codewords. This leads to the definition of *minimum distance* $d_{\min}$ of a code as the smallest Hamming distance between any pair of codewords. If $d_{\min} \geq 2t + 1$ and a channel makes t errors, then the received sequence differs from the transmitted codeword in t places and differs from every other codeword in $t + 1$ or more places. Therefore, the codeword closest to the received sequence is the correct codeword.

A simple block code, known as the (7,4) Hamming code is shown in Fig. 1. There are 16 binary codewords in this code; each codeword has a blocklength 7 and the minimum distance of the code is 3. Each of the 16 codewords can be

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

FIG. 1 Hamming (7,4) code.

assigned to represent 1 of the 16 4-bit binary datawords. Therefore, the code represents 4-bit datawords ($k = 4$) by 7-bit codewords ($n = 7$). Any one-to-one assignment of datawords to codewords can be used. A natural choice is to assign codewords so that the first four bits of the codeword are identical to the four bits of the dataword. Examination of Fig. 1 shows that this choice is consistent with the structure of the (7,4) Hamming code by choosing the first four code bits as data bits. This kind of encoder is called a *systematic encoder.* The three bits other than the four data bits are then called *parity-check* bits.

Notice that there is a careful distinction between the notion of a code and the notion of the encoder. The code shown in Fig. 1 is simply a set of codewords and is independent of the method of encoding. The term *encoder* refers to the assignment of datawords to codewords. The term also applies to the mechanism for implementing this assignment.

The code is used as follows. Four bits of data are encoded into a 7-bit codeword. The channel makes at most 1 bit error. Because the minimum distance of the code is 3 (which can be verified by inspection of Fig. 1), the received word differs from the true codeword in at most 1 bit position and differs from every other codeword in at least 2 bit positions. The decoder can easily deduce the correct codeword and then recover the 4 data bits.

Convolutional Codes

An encoder for a tree code encodes a stream of data symbols into a stream of codeword symbols. The duration of the datastream is so long that, effectively, it is infinite and does not enter into the design of the encoder and decoder. Beginning at time zero and continuing indefinitely into the future, a data sequence is shifted into the encoder and a code sequence is shifted out. A tree code breaks the datastream into segments called data frames each consisting of k data symbols, k being a small integer. The encoder is a finite-state machine that retains some memory of earlier data frames; in the simplest case, it simply stores the m most recent data frames unchanged. A single code frame consisting of n symbols is computed from the mk data symbols of the m frames stored in memory and the k symbols in the incoming data frame; these n symbols are shifted out to the channel as the k new data symbols are shifted into the encoder. The ratio k/n is called the code rate of the tree code and is denoted by R. The code frames are concatenated to form the codestream for passage through the channel. Tree codes with a special memory and linearity structure, which is defined below, are called *convolutional codes.* These are the most common types of tree codes used in practice.

The *constraint length* ν of a convolutional code is defined as the number of memory cells in a minimum encoder. The minimum distance $d_{\min}$ of a convolutional code is defined as the minimum Hamming distance between any two codewords.

One way to describe a convolutional code is by a polynomial representation. We describe a simple binary convolutional code with $k = 1$, $n = 2$, and $\nu = 2$. Let

$$a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots$$

denote the infinite sequence of binary data; each coefficient is either a 0 or a 1. Let

$$g_0(x) = x^2 + x + 1$$
$$g_1(x) = x^2 + 1.$$

These two polynomials are called *generator polynomials*. The encoder is described by the polynomial product (with modulo-2 arithmetic in the coefficients)

$$c_0(x) = g_0(x)a(x)$$
$$c_1(x) = g_1(x)a(x).$$

These polynomial products produce the two codeword polynomials

$$c_0(x) = c_{00} + c_{01}x + c_{02}x^2 + c_{03}x^3 + \ldots$$
$$c_1(x) = c_{10} + c_{11}x + c_{12}x^2 + c_{13}x^3 + \ldots$$

The binary coefficients of $c_0(x)$ and $c_1(x)$ are interleaved to form the binary codestream and sent to the channel. Thus, there are two code bits for every data bit and the code rate is 0.5.

The minimum distance of this code is 5. Two code sequences at Hamming distance 5 are defined by the data polynomial

$$a(x) = 0$$

corresponding to code polynomials

$$c_0(x) = 0$$
$$c_1(x) = 0$$

and the data polynomial

$$a(x) = 1$$

corresponding to code polynomials

$$c_0(x) = x^2 + x + 1$$
$$c_1(x) = x^2 + 1.$$

The Hamming distance between these two codestreams is clearly 5. It is easy to check that no two codestreams are closer and, consequently, the minimum distance of this code is 5. Because $2 < d_{\min}/2$, this convolutional code can correct two errors. Furthermore, this convolutional code can correct multiple error events, each having one or two bit errors, provided the error events are spaced far enough apart that they do not interact in the structure of the code.

In general, a convolutional code is described by a vector of data polynomials over some field $GF(q)$

$$\mathbf{a}(x) = [a_1(x), a_2(x), \ldots, a_k(x)].$$

The k data symbols in the ℓth frame provide the coefficients of x^ℓ. The nature of the set of generator polynomials defining the convolutional code is described by the matrix

$$\mathbf{G}(x) = \begin{bmatrix} g_{11}(x) & \ldots & g_{1n}(x) \\ g_{21}(x) & \ldots & g_{2n}(x) \\ \vdots & & \\ g_{k1}(x) & \ldots & g_{kn}(x) \end{bmatrix}$$

The matrix-vector product

$$\mathbf{c}(x) = \mathbf{a}(x)\mathbf{G}(x)$$

then gives the vector of codeword polynomials

$$\mathbf{c}(x) = [c_1(x), c_2(x), \ldots, c_n(x)].$$

The n symbols forming the coefficients of x^ℓ provide the n code symbols in the ℓth frame. They are interleaved to form the codestream.

Good convolutional codes—that is, good matrices of generator polynomials $\mathbf{G}(x)$—are found by computer search. Strong algebraic constructions analogous to those described below for Reed-Solomon codes have never been discovered.

Arithmetic in a Galois Field

The most successful rules for constructing good block codes for error prevention make use of a special kind of arithmetic known as the arithmetic of Galois fields. A Galois field with q elements, denoted $GF(q)$, is an unconventional arithmetic system containing the operations of addition, subtraction, multiplication, and division defined in such a way that most familiar arithmetic and algebraic procedures remain valid. The arithmetic operations of a Galois field have the enormous advantage that there is no overflow or round-off error. Because the error-control code is used to protect bit packages but not to do real computations, it does not matter that the arithmetic rules are unconventional.

Figure 2 shows the addition and multiplication tables for several simple Galois fields. Notice that $GF(2)$ and $GF(3)$ are modulo-2 and modulo-3 arithmetic, respectively, but $GF(4)$ is *not* modulo-4 arithmetic. [Modulo-4 arithmetic cannot form a field because $2 \cdot 1 = 2 \cdot 3 \pmod 4$, so division by 2 would not behave properly in modulo-4 arithmetic.] The larger field $GF(256)$ is very important in practice because it provides a closed arithmetic structure for the

GF(2)

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

GF(3)

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

GF(4)

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

FIG. 2 Arithmetic tables for several simple Galois fields.

set of 8-bit bytes, but the addition and multiplication tables are too large to show here. Addition and multiplication tables in large fields usually are described by rules rather than tables.

We discuss only the binary power Galois fields $GF(2^m)$ in this article. The elements of the field $GF(2^m)$ can be defined as the set of m-bit bytes. Thus, $GF(2^8)$ is the set of 8-bit bytes and $GF(16)$ is the set of 4-bit bytes (or hexadecimal characters). Addition is defined simply as componentwise modulo-2 addition (bit-by-bit exclusive-or). For example, two of the elements of $GF(16)$ are 1101 and 1110. Their addition is

$$(1101) + (1110) = (0011).$$

Multiplication is more complicated to define. The multiplication in $GF(2^m)$ must be consistent with addition in the sense that the distributive law

$$(a + b)c = ac + bc$$

holds for any a, b, and c in $GF(2^m)$. Because addition is a bit-by-bit exclusive-or, this suggests that multiplication should have the structure of a shift and exclusive-or rather than the conventional structure of shift and add.

We may try to define multiplication in accordance with the following product:

$$\begin{array}{r} 1\,1\,1\,0 \\ 1\,1\,0\,1 \\ \hline 1\,1\,1\,0 \\ 0\,0\,0\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ \hline 1\,0\,0\,0\,1\,1\,0 \end{array}$$

However, the arithmetic system must be closed under multiplication. The product of two m-bit numbers must produce an m-bit number; the wordlength does not increase. If a shift and exclusive-or structure is the right definition for the multiplier, then we also need a rule for folding back the overflow bits into the m-bits of the product. The trick is to define the overflow rule so that $b = c$ whenever $ab = ac$ and a is nonzero. Otherwise, division could not be defined and the arithmetic system would not be satisfactory.

The overflow rule is constructed in terms of polynomial division. Let $p(x)$ be an irreducible polynomial over $GF(2)$ of degree m. This means that $p(x)$ can have only coefficients equal to 0 or 1, and that $p(x)$ cannot be factored into the product of two such polynomials over $GF(2)$. Factoring $p(x)$ means writing

$$p(x) = p^{(1)}(x)p^{(2)}(x)$$

where polynomial multiplication uses modulo-2 arithmetic on the coefficients.

Multiplication of two elements a and b in $GF(2^m)$ to produce the element $c = ab$ can be described as a polynomial multiplication modulo the irreducible polynomial $p(x)$. Let a and b be numbers in $GF(2^m)$. These are m-bit binary numbers with the binary representations

$$a = (a_0, \ldots, a_{m-1})$$
$$b = (b_0, \ldots, b_{m-1}).$$

They also have the polynomial representations

$$a(x) = \sum_{i=0}^{m-1} a_i x^i$$

$$b(x) = \sum_{i=0}^{m-1} b_i x^i$$

where a_i and b_i are the ith bits of a and b, respectively. Then the product is defined as the sequence of coefficients of the polynomial

$$c(x) = a(x)b(x) \qquad \text{mod } p(x).$$

Because the coefficients are added and multiplied by the bit operations of $GF(2)$, the polynomial product is equivalent to the shift and exclusive-or operations mentioned above. The modulo-$p(x)$ operation specifies the rule for folding overflow bits back into the m-bits of the field element. Simply divide the product polynomial by the polynomial $p(x)$ and keep the remainder. With this definition of multiplication and the definition of addition noted above, the description of the Galois field $GF(2^m)$ is complete.

As an example of a Galois field, we construct $GF(2^4)$. The 16 elements are the set of 4-bit bytes

$$GF(2^4) = \{0000, 0001, 0010, \ldots, 1111\}$$

and addition of two elements is bit-by-bit modulo-2 addition. To define multiplication, we use the polynomial

$$p(x) = x^4 + x + 1.$$

To verify that this polynomial is irreducible, we can check that x, $x + 1$, $x^2 + 1$, $x^2 + x + 1$ are not factors, whereas $p(x)$ must have either a first-degree factor or a second-degree factor if it is reducible.

Then, to multiply 0101 by 1011, for instance, we represent these by $a(x) = x^2 + 1$ and $b(x) = x^3 + x + 1$ and write

$$\begin{aligned} c(x) &= (x^2 + 1)(x^3 + x + 1) \qquad \bmod p(x) \\ &= x^5 + x^2 + x + 1 \qquad \bmod x^4 + x + 1 \end{aligned}$$

The modulo-$p(x)$ operation consists of division by $p(x)$ and keeping the remainder polynomial. Carrying out the division for the sample calculation gives

$$c(x) = 1$$

which is the polynomial representation for binary 0001, giving the product in $GF(16)$

$$(0101)(1011) = (0001).$$

Because the product happens to be equal to 1, this example also tells us how to divide in $GF(16)$. We see that

$$(0101)^{-1} = (1011)$$

because $(0101)^{-1}$ is defined to be the field element for which $(0101)^{-1}(0101) = 1$. Likewise

$$(1011)^{-1} = (0101).$$

To divide by 0101 we multiply by 1011, while to divide by 1011 we multiply by 0101. For example

$$\frac{(0110)}{(1011)} = (0110)(0101)$$
$$= (1101)$$

where the product is calculated as described above.

Division will be possible for every nonzero element a if a^{-1} exists. The inverse a^{-1} is the value of b that solves

$$ab = 1.$$

This equation always has a solution (except when $a = 0$) if $p(x)$ is chosen to be an irreducible polynomial, so the use of an irreducible polynomial ensures that division is defined. The reason why this is so is quite involved and is not explained further.

Although the multiplication and division rules of a Galois field may appear unfamiliar to us, logic circuits or computer subroutines to implement them are straightforward. One could even build a programmable computer with Galois-field arithmetic as primitive instructions.

Algebraic manipulations in the Galois field $GF(2^m)$ behave very much like algebraic manipulations in the fields more usually encountered in engineering problems such as the real field or the complex field. The conventional algebraic properties of associativity, commutativity, and distributivity all hold. Methods of solving linear systems of equations are valid, including matrix algebra, determinants, and so forth. There is even a discrete Fourier transform in $GF(2^m)$ and it has all the familiar properties of the discrete Fourier transform. The Fourier transform is particularly important to our purposes because it is the basis of the definition of the Reed-Solomon code.

Hamming Codes on Bytes

A simple application of Galois fields is to construct those single-byte-error-correcting codes known as Hamming codes. In the alphabet of m-bit bytes, the codes are constructed with the arithmetic of $GF(2^m)$. For each value of r, it is possible to construct a Hamming code for $GF(2^m)$ with r parity bytes and blocklength $n = (2^{mr}-1)/(2^m-1)$. Thus, each codeword can encode $k = [(2^{mr}-1)/(2^m-1)]-r$ data bytes. For example, for $m = 8$ and $r = 2$, the code symbols are 8-bit bytes; there are $r = 2$ parity-check symbols, the blocklength n equals 257, and there are $k = 255$ data symbols. This Hamming code can be shortened by one symbol (by permanently setting one data symbol to 0 mathematically, and thereafter deleting it from the codeword) to produce a (256,254) Hamming code over 8-bit bytes. Such a code could be used to find and correct 1 byte error in a block of 256 8-bit bytes.

The values n and k for some Hamming codes are given in Fig. 3. Any of these codes can be shortened to a convenient blocklength by deleting data symbols.

The construction is simple to describe when $r = 2$. For example, we describe

GF(2)	*GF*(4)	*GF*(16)	*GF*(256)
(7,4)	(5,3)	(17,15)	(257,255)
(15,11)	(21,18)	(273,270)	(65793,65790)
(31,26)	(85,81)		
(63,57)	(341,336)		
(127,120)			

FIG. 3 Parameters (n,k) for some Hamming codes.

the code over $GF(16)$ with (hexadecimal) data symbols $(a_1, a_2, \ldots, a_{15})$. Write the parity-check symbols as

$$p_1 = a_1 + a_2 + a_3 + \ldots + a_{14} + a_{15}$$
$$p_2 = a_1 + 2a_2 + 3a_3 + \ldots + Ea_{14} + Fa_{15}.$$

The 15 coefficients in p_2 are the 15 nonzero elements of $GF(16)$. The equations can be abbreviated as

$$p_1 = \Sigma_i a_i$$

$$p_2 = \Sigma_i g_i a_i$$

where g_i is the field element pointing at the ith component. After every block of 15 (hexadecimal) data symbols, these 2 (hexadecimal) parity-check symbols are inserted to form the codeword $(c_1, c_2, \ldots, c_{15}, c_{16}, c_{17}) = (a_1, a_2, \ldots, a_{15}, p_1, p_2)$.

The decoder can correct a single symbol error in the block of 17 symbols as follows. The received word is $(v_1, v_2, \ldots, v_{15}, v_{16}, v_{17})$, the first 15 symbols correspond to data symbols, the last 2 symbols correspond to parity-check symbols, and at most 1 symbol is in error. Let e_i denote the error in component i. Then

$$v_i = c_i + e_i.$$

Compute the following terms, known as *syndromes*

$$s_1 = v_1 + v_2 + v_3 + \ldots + v_{14} + v_{15} - v_{16}$$
$$s_2 = v_1 + 2v_2 + 3v_3 + \ldots + Ev_{14} + Fv_{15} - v_{17}$$

If both s_1 and s_2 equal 0, there is no error. If either s_1 or s_2 equals 0, the single error occurs in a parity-check symbol. If both s_1 and s_2 are nonzero, the single error e_i occurred in a data symbol, and

$$s_1 = e_i$$

$$s_2 = g_i e_i.$$

Therefore, s_1 gives the byte value of the error and $s_2/s_1 = g_i$ points to the ith component, the component that is in error.

Such a procedure with $r = 2$ applies to any field $GF(2^m)$ to give a code with $2^m - 1$ data bytes and 2 parity-check bytes. When the number of data bytes is larger than $2^m - 1$, there are not enough field elements for one field element to point to a unique component. That is why more than 2 parity-check bytes are then necessary.

Reed-Solomon Codes and BCH (Bose-Chaudhuri-Hocquenghem) Codes

The most important block codes in applications are those codes known as Reed-Solomon codes and the closely related codes known as BCH (Bose-Chaudhuri-Hocquenghem) codes. The Reed-Solomon codes are codes on the byte level, while the BCH codes are (usually) codes on the bit level.

The Reed-Solomon codes are defined using the mathematics of the discrete Fourier transform in a Galois field. This Fourier transform is defined in exactly the same way as the Fourier transform in the complex field. Specifically, if $\mathbf{v}$ is a vector of blocklength n of symbols from the field $GF(q)$, then the Fourier transform of $\mathbf{v}$ is another vector, denoted $\mathbf{V}$, also of blocklength n, given by

$$V_j = \sum_{i=0}^{n-1} \omega^{ij} v_i \qquad j = 0, \ldots, n-1$$

where ω is an element of $GF(q)$ of order n. That is, $\omega^n = 1$ and $\omega^k \neq 1$ if $k < n$. Notice that this is a close analog of the discrete Fourier transform in the complex field.

$$V_k = \sum_{i=0}^{n-1} (e^{-j2\pi/n})^{ik} v_i \qquad k = 0, \ldots, n-1$$

The only qualification that needs to be made is that a Fourier transform of blocklength n exists in the Galois field $GF(q)$ only if $GF(q)$ contains an element ω of order n. Only if n divides $q - 1$ does such an ω exist. In particular, if q is a power of 2, n is always odd. For example, in $GF(256)$, n can only take the values 255, 85, 51, 17, 15, 5, and 3.

The Fourier transform in a Galois field has all of the properties that one might expect and they can be proved in the natural way. Specifically, there is a modulation/delay theorem that says that if $v'_i = v_i \omega^r$, then $V'_j = V_{j-r}$, a convolution theorem that says that if $v_i = u_i w_i$, then

$$V_j = \sum_{k=0}^{n-1} U_k W_{((j-k))},$$

an inverse Fourier transform, and so on. The inverse Fourier transform has the usual form

$$v_i = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{-ij} V_j$$

where n in the denominator multiplying the sum is interpreted as an integer of $GF(q)$. That is, n is the sum of 1 with itself n times. Because $1 + 1 = 0$ in $GF(2^m)$ for any m, and n is odd, the integer n in the Galois field $GF(2^m)$ is equal to 1 and can be suppressed in the equation for the inverse Fourier transform.

The t-error-correcting Reed-Solomon code of blocklength n is the set of all vectors $\mathbf{c}$ of blocklength n whose spectrum satisfies $C_j = 0$ for $j = n - 2t, \ldots, n - 1$. This code is described briefly as an $(n, n - 2t)$ Reed-Solomon code.

One way to find the Reed-Solomon codewords is to encode in the frequency domain as is suggested by the definition. This means setting $C_j = 0$ for $j = n - 2t, \ldots, n - 1$, and setting the remaining $n - 2t$ components of the transform equal to the $n - 2t$ data symbols given by $a_0, \ldots, a_{n-2t-1}$. That is

$$C_j = \begin{cases} a_j & j = 0, \ldots, n - 2t - 1 \\ 0 & j = n - 2t, \ldots, n - 1 \end{cases}$$

An inverse Fourier transform then produces the codeword $\mathbf{c}$. The number of data symbols encoded equals $n - 2t$ and there are $2t$ extra symbols in the codeword to correct t errors. A Reed-Solomon code always uses two overhead symbols for every error to be corrected.

Using the inverse Fourier transform is not the only way to encode the $n - 2t$ data symbols into the codewords—others may yield a simpler implementation—but the frequency-domain encoder is the most instructive because it exhibits very explicitly the notion that the codewords are those vectors with the same set of $2t$ zeros in the transform domain.

An alternative encoder in the time domain works as follows. The $n - 2t$ data symbols are expressed as a polynomial

$$a(x) = a_{n-2t-1}x^{n-2t-1} + a_{n-2t-2}x^{n-2t-2} + \ldots + a_1x + a_0$$

where $a_0, a_1, \ldots, a_{n-2t-1}$ are the $n - 2t$ data symbols. Then the n codeword symbols are given by the coefficients of the polynomial product

$$c(x) = g(x)a(x)$$

where $g(x)$ is a fixed polynomial called the *generator polynomial*. The generator polynomial is the unique monic (leading coefficient equals 1) polynomial of degree $2t$ that has zeros at $\omega^{n-2t}, \omega^{n-2t+1}, \ldots, \omega^{n-1}$. It can be obtained by multiplying out the expression

$$g(x) = (x - \omega^{n-2t})(x - \omega^{n-2t+1}) \ldots (x - \omega^{n-1}).$$

We can verify that this time-domain encoder does indeed produce a Reed-Solomon code. The Fourier transform of the codeword is formally the same as evaluating the polynomial $c(x)$ at ω^j. That is,

$$C_j = \sum_{i=0}^{n-1} c_i \omega^{ij} = c(\omega^j) = g(\omega^j)a(\omega^j).$$

By the definition of $g(x)$, $g(\omega^j)$ equals 0 for $j = n - 2t, \ldots, n - 1$. Consequently, $C_j = 0$ for $j = n - 2t, \ldots, n - 1$. Therefore, the encoding in the time domain does produce legitimate Reed-Solomon codewords. The set of codewords produced by the time-domain encoder is the same as the set of codewords produced by the frequency-domain encoder, but the mapping between datawords and codewords is different.

Thus far, both methods of encoding discussed have the property that the symbols of the dataword do not appear explicitly in the codeword. Another method of encoding, known as *systematic encoding*, leaves the data symbols unchanged and contained in the first $n - 2t$ components of the codeword. Multiplication of $a(x)$ by x^{2t} will move the components of $a(x)$ left $2t$ places. Thus, we can write an encoding rule as

$$c(x) = x^{2t}a(x) + r(x)$$

where $r(x)$ is a polynomial of degree less than $2t$ appended to make the spectrum be a legitimate codeword spectrum. The spectrum will be correct if $c(x)$ is a multiple of $g(x)$ and this will be so if $r(x)$ is chosen as the negative of the remainder when $x^{2t}a(x)$ is divided by $g(x)$. Thus, because it gives a multiple of $g(x)$, the equation

$$c(x) = x^{2t}a(x) - R_{g(x)}[x^{2t}a(x)]$$

defines a systematic encoder where the operator $R_{g(x)}$ takes the remainder under division by $g(x)$. This definition of $c(x)$ is indeed a codeword because it has zero remainder when divided by $g(x)$. Thus

$$\begin{aligned} R_{g(x)}[c(x)] &= R_{g(x)}[x^{2t}a(x)] - R_{g(x)}\{R_{g(x)}[x^{2t}a(x)]\} \\ &= 0 \end{aligned}$$

because remaindering can be distributed across addition.

A decoder for a Reed-Solomon code does not depend on how the codewords are used to store information except for the final step of reading the data symbols out of the corrected codeword.

To prove that an $(n,n - 2t)$ Reed-Solomon code can correct t symbol errors, it is enough to prove that every two codewords in the code differ from each other in at least $2t + 1$ places. If this is true, then changing any t components of a codeword will produce a word that is different from the correct codeword in t components and is different from every other codeword in at least $t + 1$ components. If, at most, t errors occur, then choosing the codeword that differs from the noisy received word in the fewest number of components will recover the correct codeword. If each symbol is more likely to be correct than to be in error, then choosing the codeword that differs from the noisy received word in the fewest places will recover the most likely codeword and will minimize the probability of decoding error.

By definition of the code,

$$C_j = 0 \qquad j = n - 2t, n - 2t + 1, \ldots, n - 1.$$

By linearity of the Fourier transform, the difference in two codewords (computed componentwise) then must also have a spectrum that is zero for $j = n - 2t, \ldots, n - 1$ and so itself is a codeword. We only need to prove that no codeword has fewer than $2t + 1$ nonzero components unless it is zero in every component. Let

$$C(y) = \sum_{j=0}^{n-2t-1} C_j y^j.$$

This is a polynomial of degree at most $n - 2t - 1$, so, by the fundamental theorem of algebra, it has at most $n - 2t - 1$ zeros. Therefore,

$$\begin{aligned} c_i &= \frac{1}{n} \sum_{j=0}^{n-1} \omega^{-ij} C_j \\ &= \frac{1}{n} C(\omega^{-i}) \end{aligned}$$

can be zero in at most $n - 2t - 1$ places, and so it is nonzero in at least $2t + 1$ places. Therefore, we can conclude that an $(n, n - 2t)$ Reed-Solomon code can correct t symbol errors.

Future Developments

A code is judged by its rate R, minimum distance $d_{\min}$, and blocklength n. For any fixed rate R, which determines the percentage of coding overhead, the best code is the one with the largest value of $d_{\min}$, which determines the performance of the code. The blocklength n is also important. A code of blocklength $10n$ that protects against $10t$ channel errors is much better than 10 uses of a code of blocklength n that protects against t errors per block, because the latter code requires the errors to be distributed with at most t errors to a block.

The block codes and tree codes that are known and in use have shown their worth in many applications. One may naturally ask whether better codes are possible. Remarkably, we know that, when the blocklength is large, the known codes are not nearly as good as the best possible codes. Indeed, if messages longer than a few thousand bits are to be transmitted, the known codes can be surprisingly weak in comparison to the best codes. However, despite more than 40 years of intense effort, there is little known about how to find optimum codes, nor is much known about good encoding and decoding algorithms for such codes.

Let $\delta = d_{\min}/n$ denote the fractional minimum distance of the code. The code can prevent decoded errors even if a fraction $\frac{1}{2}\delta$ of the channel symbols are in error. It is known that if the blocklength is large enough, codes exist that have R and δ satisfying (within ϵ) the equation

$$R = 1 + \delta \log_2 \delta + (1 - \delta) \log_2 (1 - \delta)$$

which is known as the *Gilbert bound* and is shown in Fig. 4. For example, with $R = 0.5$, $\delta = 0.11$. Thus, if one allows that 50% of channel bits are overhead, a code exists that will decode correctly even if any 5.5% of the channel bits are in error. With such a code, were it known, a message codeword of 1 million code bits would be received correctly if any 55,000 channel bits were in error, for instance the first 55,000. Similarly, with $R = 0.9$, $\delta = 0.013$. If one allows that 10% of channel bits are overhead, a code exists that will decode correctly even if any 0.65% of channel bits are in error. With such a code, were it known, a message codeword of 1 million code bits would be received correctly even if any 6500 bits were in error.

Unfortunately, though we know that it is possible in principle to do at least as well as the Gilbert bound, we have no idea how to construct such codes. Furthermore, we do not even know that the Gilbert bound is a tight bound; it may be possible to construct binary codes that are even better than the Gilbert bound.

Attempts to find the codes of exceptional performance promised by the Gilbert bound are continuing and are likely to succeed eventually. The major effort employs the tools of algebraic geometry and was launched by the discovery of the *Goppa codes*. There has also been slight progress in the development of nonlinear block codes launched by the discovery of the *Preparata codes*.

Algorithms for Decoding

A practical decoder for an error-control code cannot compare exhaustively the received word to every codeword to see which codeword most closely agrees

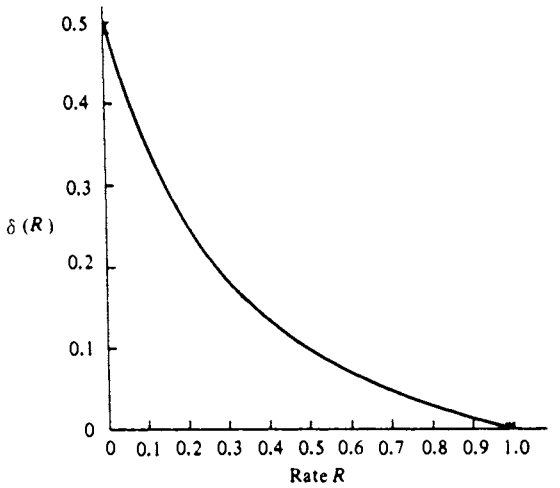

FIG. 4 Gilbert bound.

with the received word. This is because the number of codewords in many codes is astronomical. For example, the (32,28) code over $GF(256)$ used by the compact disk has 256^{28} (about 2.69×10^{67}) codewords. A decoder for this code is practical only if it uses some algorithmic procedure that computes the correct codeword from the received word.

Strong algorithms, such as the *Berlekamp-Massey algorithm*, for decoding Reed-Solomon codes are known and are in use. This algorithm can be described using the language of spectral estimation, but applied in a Galois field.

The best algorithm for decoding convolutional codes, known as the *Viterbi algorithm*, is essentially an efficient, computational algorithm for searching all possible codewords exhaustively.

Algorithms such as the Berlekamp-Massey algorithm for decoding Reed-Solomon codes have an essentially quadratic computational cost in the performance of the code. Algorithms such as the Viterbi algorithm for decoding convolutional codes have an essentially exponential computational cost in the performance of the code.

Spectral Estimation in a Finite Field

A practical decoder for a Reed-Solomon code requires a computational procedure to find the transmitted codeword. The codeword $\mathbf{c}$ is transmitted and the channel makes an error described by the vector $\mathbf{e}$, which is nonzero in not more than t places. The received word $\mathbf{v}$ is written componentwise as

$$v_i = c_i + e_i \qquad i = 0, \ldots, n-1.$$

The decoder must process the received word $\mathbf{v}$ to remove the error word $\mathbf{e}$; the dataword is then recovered from $\mathbf{c}$.

The received noisy codeword $\mathbf{v}$ has a Fourier transform

$$V_j = \sum_{i=0}^{n-1} \omega^{ij} v_i \qquad j = 0, \ldots, n-1$$

with components $V_j = C_j + E_j$ for $j = 0, \ldots, n-1$. But, by construction of a Reed-Solomon code,

$$C_j = 0 \qquad j = n - 2t, \ldots, n-1.$$

Hence

$$V_j = E_j \qquad j = n - 2t, \ldots, n-1.$$

This block of $2t$ components of $\mathbf{V}$ gives us a window through which we can look at $2t$ of the n components of the error-pattern transform. The decoder must find all n components of $\mathbf{E}$ given a segment consisting of $2t$ components

of **E**, and the additional information that, at most, t components of the time-domain error pattern **e** are nonzero. Once **E** is known, the computation is trivial because $C_j = V_j - E_j$. From **C** one can compute **c**. The data symbols are recovered easily in various ways, depending on the method of encoding.

To find a procedure to compute **E**, we use properties of the Fourier transform. Suppose, for the moment, that there exists a polynomial $\Lambda(x)$ of degree at most t and with $\Lambda_0 = 1$ such that

$$\Lambda(x)E(x) = 0 \qquad (\text{mod } x^n - 1)$$

where

$$E(x) = \sum_{j=0}^{n-1} E_j x^j.$$

The polynomial product is equivalent to a cyclic convolution. Using the assumed properties of $\Lambda(x)$ we can rewrite the convolution as

$$\Lambda_0 E_j + \sum_{k=1}^{t} \Lambda_k E_{j-k} = 0$$

or, because $\Lambda_0 = 1$,

$$E_j = -\sum_{k=1}^{t} \Lambda_k E_{j-k} \qquad j = 0, \ldots, n-1.$$

This equation may be recognized as a description of a kind of filter know as an *autoregressive filter* with t taps. It defines component E_j in terms of the preceding t components $E_{j-1}, \ldots, E_{j-t}$. Since we know $2t$ components of **E**, by setting $j = n - t, \ldots, n - 1$, we can write down the following set of $n - 1$ equations:

$$\begin{aligned}
E_{n-t} &= -\Lambda_1 E_{n-t-1} - \Lambda_2 E_{n-t-2} - \cdots - \Lambda_t E_{n-2t} \\
E_{n-t+1} &= -\Lambda_1 E_{n-t} - \Lambda_2 E_{n-t-1} - \cdots - \Lambda_t E_{n-2t+1} \\
&\vdots \\
E_{n-1} &= -\Lambda_1 E_{n-2} - \Lambda_2 E_{n-3} - \cdots - \Lambda_t E_{n-t-1}
\end{aligned}$$

Here, there are t equations, linear in the unknown components Λ_k and involving only known components of **E**. Hence, provided there is a solution, we can find $\boldsymbol{\Lambda}$ by solving this system of linear equations. The solution will give the polynomial $\Lambda(x)$ that was assumed earlier. Once $\Lambda(x)$ is known, all other values of **E** can be obtained by recursive computation using the equation

$$E_j = -\sum_{k=1}^{t} \Lambda_k E_{j-k} \qquad j = 0, \ldots, n - 2t - 1$$

recalling that the indices are modulo-n.

To verify that the system of equations has a solution, suppose that there are $\nu \leq t$ nonzero errors at locations with indices i_ℓ for $\ell = 1, \ldots, \nu$. Define the polynomial $\Lambda(x)$, known as the *error-locator polynomial*, by

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 - x\omega^{i_\ell})$$

which has at most degree t and $\Lambda_0 = 1$. The vector $\mathbf{\Lambda}$ of length n whose components Λ_j are coefficients of the polynomial $\Lambda(x)$ has an inverse transform

$$\begin{aligned}\lambda_i &= \frac{1}{n} \sum_{j=0}^{n-1} \Lambda_j \omega^{-ij} = \frac{1}{n} \Lambda(\omega^{-i}) \\ &= \frac{1}{n} \prod_{\ell=1}^{\nu} (1 - \omega^{-i}\omega^{i_\ell})\end{aligned}$$

which is zero if $i = i_\ell$. Therefore, $\lambda_i e_i = 0$ for all i. Because a product in the time domain corresponds to a cyclic convolution in the frequency domain, we see that the cyclic convolution in the frequency domain is equal to zero.

$$\mathbf{\Lambda} * \mathbf{E} = \mathbf{0}.$$

Hence, a polynomial $\Lambda(x)$ solving $\Lambda(x)E(x) = 0 \pmod{x^n - 1}$ does exist.

The Berlekamp-Massey Algorithm

The procedure for decoding Reed-Solomon codes requires solution of the matrix-vector equation

$$\begin{bmatrix} E_{n-t-1} & E_{n-t-2} & \cdots & E_{n-2t} \\ E_{n-t} & E_{n-t-1} & \cdots & E_{n-2t+1} \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ E_{n-2} & E_{n-3} & \cdots & E_{n-t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \cdot \\ \cdot \\ \cdot \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} E_{n-t} \\ E_{n-t+1} \\ \cdot \\ \cdot \\ \cdot \\ E_{n-1} \end{bmatrix}$$

The matrix is a special kind of matrix known as a *Toeplitz matrix* because the elements in any subdiagonal are equal. The computational problem is one of inverting a Toeplitz system of equations. This kind of computational problem arises frequently in digital-signal processing not only in decoding Reed-Solomon codes, but also in the design of autoregressive filters and in spectral analysis. To invert a Toeplitz system of equations, special computational algorithms usually are used because they are computationally more efficient than computing the matrix inverse by a general method.

The Berlekamp-Massey algorithm is the preferred method for solving this system of equations in a Galois field. The algorithm is easy to execute, though

it is difficult to derive. At the rth of $2t$ iterations, the algorithm computes a shortest-length autoregressive filter with length L_r and taps $\Lambda_i^{(r)}$ that satisfies

$$E_j = -\sum_{i=1}^{L_r} \Lambda_i^{(r)} E_{j-i}$$

for $j = 1, \ldots, r$. The shift register for the $(r + 1)$th iteration is then computed from the shift register for the rth iteration. There are $2t$ iterations and each has computational complexity t. Thus, the algorithm has computational complexity proportional to t^2. The Berlekamp-Massey algorithm can be implemented easily either in hardware or in software.

The Viterbi Algorithm

The most important method of decoding convolutional codes is the Viterbi algorithm. The computational complexity of this algorithm is exponential in the constraint length so it can be used only for convolutional codes of small constraint length. The algorithm is best described as an efficient method for searching a trellis.

A *trellis* is a graphical representation of the memory that exists in many kinds of data sequences, in particular the codestream of a convolutional code. An example of a very small trellis is shown in Fig. 5. Each node in one column denotes one possible state of a finite-state machine. Each column denotes a different time frame. In a practical problem of decoding a convolutional code of constraint length ν, there are 2^ν nodes in each column and the trellis extends indefinitely to the right.

A convolutional code generates a sequence of numbers by starting at the left-most node of the trellis and moving to the right along any path, one node per unit of time. A time unit might be as small as 100 nanoseconds, so the source might move through 10 million nodes per second. As the encoder runs through the trellis, it encounters a series of labels that then become the

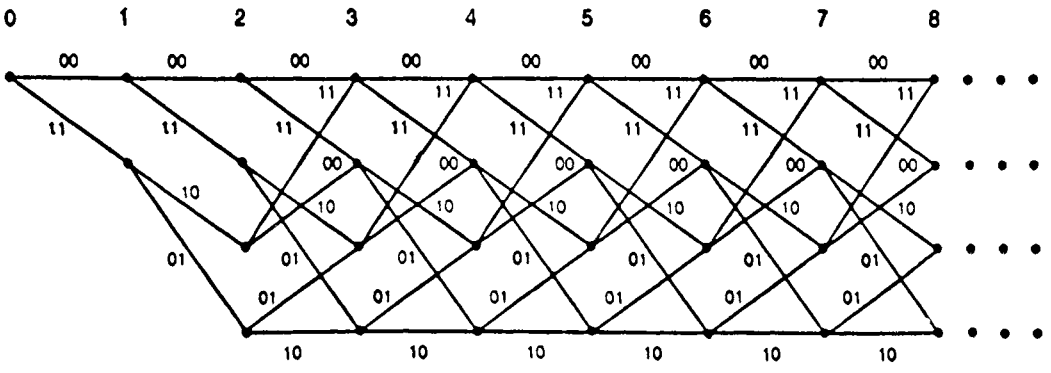

FIG. 5 A simple trellis.

codeword symbols. By observing the channel output, the decoder must deduce which path the encoder took through the trellis. The Viterbi algorithm finds the path whose symbol sequence best agrees with the received sequence in the sense of minimum Hamming distance.

In principle, it is easy to find the best path. Simply lay the received sequence along every possible path through the trellis, compute the Hamming distance to that codeword, and choose the path whose Hamming distance is smallest. The Viterbi algorithm is a recursive procedure to do this computation based on the general principle that if the optimal path from point A to point C passes through point B, then that section of the path from point B to point C must correspond to the optimal path from point B to point C.

The Viterbi algorithm operates frame-by-frame tracing through the trellis. An iteration begins with the assumption that the minimum-distance path to every node in that frame is known. Then, in the next frame, the algorithm determines the most likely path to each of the new nodes. To get to any one of the nodes in the new frame, the path must pass through one of the nodes in the previous frame. One can get the candidate paths to a new node by extending to this new node each of the old paths that can be so extended. The minimum-distance path is found simply by updating a table of possible paths and choosing the best path to the new node. This process is repeated for each of the nodes in the new frame. The algorithm continues in this way, computing frames indefinitely.

Performance on Noisy Channels

Codes for the prevention of errors are used to communicate over noisy channels. When evaluated for a specific channel, a code is judged by its rate and by its probability of decoding failure. By the nature of noise, errors are randomly distributed. There is always some possibility that more errors will occur than the code can correct. Then the decoder fails on that block or on that segment of the codestream. There are two kinds of decoding failure: a decoding default and a decoding error. When the decoder defaults, it announces to the user that a portion of the message is bad and cannot be corrected. It does not give the user bad data as if it were correct. This is the less-serious failure mode.

The decoder may also fail by "correcting" the wrong symbols in a bad message, thereby making the message even worse and passing the message to the user as a good message. This is a decoding error. In most well-designed systems, the probability of decoding error is many orders of magnitude smaller than the probability of decoding default.

Coding Gain

A systems designer may elect to use a code not to reduce bit error rate, but rather to enable the system to operate at a lower signal-to-noise ratio at the

same bit error rate. The reduction in the required signal-to-noise ratio obtained by coding is called *coding gain*. For example, a simple binary phase-shift-keyed (BPSK) communication system operates at a bit error rate of 10^{-5} at a signal-to-noise ratio (E_b/N_0) of 9.6 dB on an additive Gaussian-noise channel. By adding a sufficiently strong error-control code to the communication system, the ratio E_b/N_0 could be reduced. If the code requires only 6.6 dB for a bit error rate of 10^{-5}, then we say that the code has a coding gain of 3 dB at a bit error rate of 10^{-5}. It is important to emphasize that this coding gain is just as real as doubling the size of an antenna or increasing the transmitted power by 3 dB. Because it is usually far easier to obtain coding gain than to increase received power, the use of a code is preferred.

In most applications, the average transmitted power is fixed. Parity-check symbols can be inserted into the channel codestream only by taking power from the data symbols. Therefore, coding gain is defined in terms of required E_b/N_0. The energy-per-bit E_b is defined as the total energy in the message divided by the number of data bits. This is equal to the energy received per channel bit divided by the code rate. The channel spectral-noise density is denoted N_0.

Figure 6 illustrates the probability of error versus E_b/N_0 for a simple binary code known as a Golay code. The coding gain at a bit error rate of 10^{-5} is 2.1 dB. For a larger code, the coding gain will be larger. The discussion, however, will be more complicated if one makes a careful distinction between decoding errors and decoding defaults.

Arbitrarily large coding gains are not possible. The illustration in Fig. 7 indicates the region where a coding gain must lie. There is one region for a hard-decision demodulator given by $E_b/N_0 > 0.4$ dB and another for a soft-decision demodulator given by $E_b/N_0 > -1.6$ dB.

Error Detection and Retransmission

A code with minimum distance $d_{\min}$ can detect every pattern of $d_{\min} - 1$ or fewer errors, but can correct only $\frac{1}{2}(d_{\min} - 1)$ or fewer errors. Furthermore, the complexity of a decoder that only detects errors is much less than a decoder that corrects errors. To detect errors when the code is systematic, one simply recomputes all the parity-check symbols from the data symbols. If any disagree with the received parity-check symbols, an error is detected.

Accordingly, many systems early on were designed only to detect errors. The errors are not corrected; rather, the transmitter is asked to retransmit the message. There are several reasons, however, why error detection and retransmission is an outdated approach. Powerful error-correction decoders are now inexpensive; on the other hand, retransmission has an adverse effect on system protocol and, accordingly, can actually be more complex than forward error correction.

Retransmission protocols require message buffering at the transmitter, a feedback link for the repeat request, and a scheme for inserting the retransmitted message into the received datastream. Normal data flow must be interrupted for the retransmission. The flow control can become extremely complex for a multi-access system, especially for one that uses relays and packetized messages.

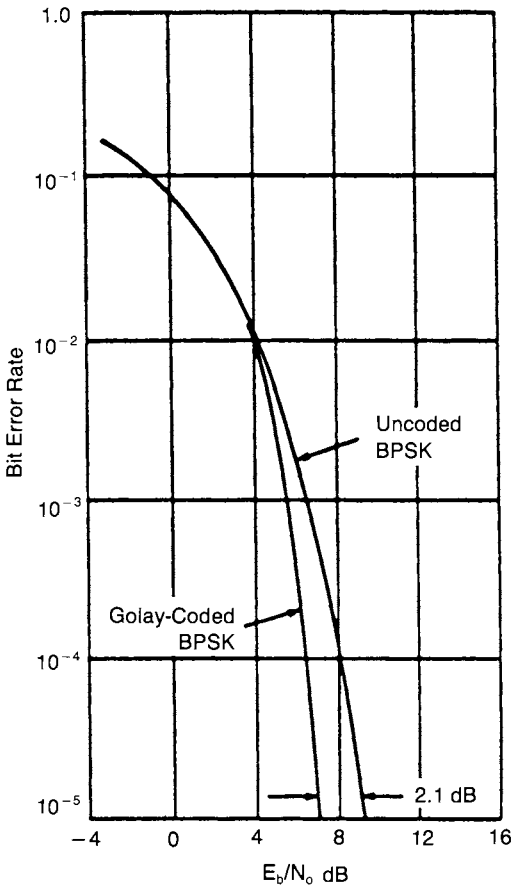

FIG. 6 Coding gain.

Another consideration is that retransmission techniques can only be used successfully if the majority of code blocks are received error-free. This is contrary to the modern trend of using the full power of a code to reduce the required signal-to-noise ratio at the receiver, or to increase spectral density of the waveform. Low-power messages will contain many errors that can be removed by error correction. Retransmission, however, only would lead to another noisy message.

Channel Capacity

Information theory tells us that every communication channel can be described concisely by a number, called the channel's *capacity* (C), that tells the data rate (in bits per second) that can be transmitted reliably over that channel. Thus, for any rate R smaller than C, one can construct a coding scheme for that channel that will prevent errors with any reliability desired; the probability of bit error can be made as small as desired.

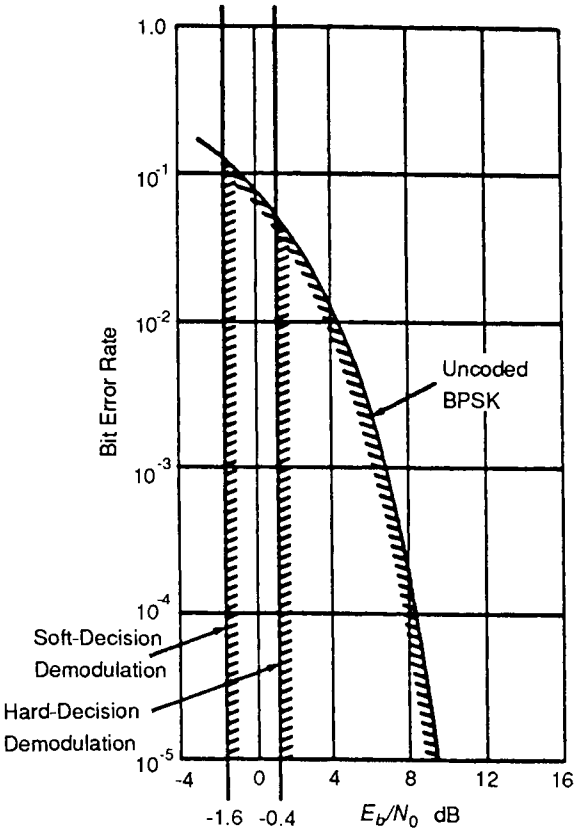

FIG. 7 Limits of coding gain.

Suppose that a binary channel makes random bit errors independently and with bit error probability ϵ. The capacity of this channel is

$$C = 1 + \epsilon \log_2 \epsilon + (1 - \epsilon) \log_2(1 - \epsilon)$$

bits per channel use. The theory says that a code exists to transmit data reliably (at an arbitrarily small probability of error) at any rate R smaller than the capacity.

For example, suppose $\epsilon = 0.11$. This means that each channel bit is wrong with a probability of 11%. Then, $C = 0.5$ bits per channel symbol. Codes of rate 0.5 exist that have any decoding-failure probability desired. However, such a code may be impractically complex if the probability is chosen very small, say 10^{-10}.

The basic theorems of information theory tell us that good codes exist. These theorems do not tell us, however, that good codes for reliable communication are simple to implement, nor do those theorems tell us how to find the good codes.

Many binary channels actually are made from an additive Gaussian-noise

channel by a BPSK modulator/demodulator (modem). We may wish to know whether the choice of modem has appreciably decreased the channel's capacity. The capacity of the original additive Gaussian-noise channel is illustrated in Fig. 8, together with the capacity of the derived binary channel. The difference in capacity between these cases is significant and was the motivation for the development of codes based on Euclidean distance, discussed in the following section.

Codes Based on Euclidean Distance

Some codes are designed to maximize the minimum Euclidean distance between any pair of codewords instead of maximizing the minimum Hamming distance. This is appropriate when the channel input and output is a real or complex number and the channel noise is additive Gaussian noise.

Given two sequences of the same length of symbols from the real or complex number system, the squared Euclidean distance between the sequences is the sum of the (magnitude) squares of the componentwise differences. The Euclidean distance between the sequences, then, is the square root of the squared Euclidean distance. For example, consider the two sequences of length 6 of real numbers given by $\mathbf{v} = (0,1,2,3,4,5)$ and $\mathbf{v}' = (0,1,4,5,4,5)$. The Euclidean distance between these sequences is

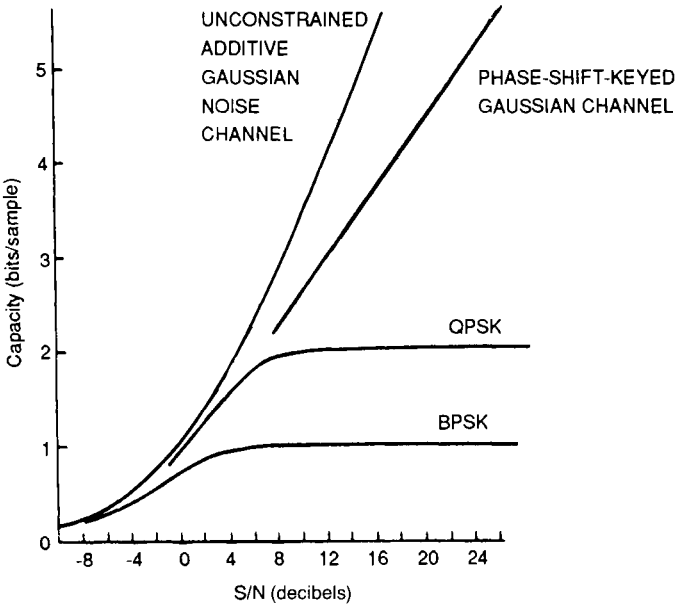

FIG. 8 Channel capacity.

$$d_E(\mathbf{v},\mathbf{v}') = \sqrt{(0-0)^2 + (1-1)^2 + (2-4)^2 + (3-5)^2 + (4-4)^2 + (5-5)^2}$$
$$= 2\sqrt{2}$$

We may also have two infinitely long sequences of real or complex numbers. Their Euclidean distance is defined in a similar way as the square root of an infinite sum of squared differences.

Soft-Decision Decoders

Every binary code (with symbols in $\{0,1\}$) can be used on the real bipolar channel (with symbols in $\{-1,1\}$). Simply represent a 0 by the real number -1 and a 1 by the real number 1. Two binary codewords are then mapped into two bipolar codewords. The squared Euclidian distance between two sequences is four times the Hamming distance

$$d_E^2(\mathbf{c},\mathbf{c}') = 4d_H(\mathbf{c},\mathbf{c}').$$

Consequently, the minimum squared Euclidean distance of the code is four times the minimum Hamming distance.

The output of the additive Gaussian-noise channel is the sequence of real random variables

$$v_i = c_i + n_i$$

where n_i is a Gaussian-noise random variable. Whereas a hard-decision demodulator estimates the binary symbols 0 or 1 from v_i, a soft-decision demodulator provides v_i as a real number directly to the decoder. The decoder is then known as a soft-decision decoder.

A soft-decision decoder calculates codeword $\mathbf{c}$ such that the Euclidean distance $d_E(\mathbf{v},\mathbf{c})$ is smallest. It is quite straightforward to modify the Viterbi algorithm to use Euclidean distance in place of Hamming distance, therefore convolutional codes are quite suitable for Gaussian channels.

Reed-Solomon codes are in a nonbinary alphabet; there is more than one way to modulate nonbinary symbols into the real (or complex) number systems, resulting in more than one way to introduce a Euclidean-distance structure. These possibilities are not discussed in this article other than to mention that the Berlekamp-Massey algorithm can be augmented to serve as a soft-decision decoder, though with some increase in complexity.

The probability of error of a soft-decision decoder can be approximated by

$$P_e \sim Q\left(\frac{d_{\min}}{2\sigma}\right)$$

where σ is the noise variance and $d_{\min}$ is the minimum Euclidean distance of the code. Clearly, increasing $d_{\min}$ by choosing a better code is as good as reducing the channel-noise variance σ. This argument shows the role of an error-prevention code in many systems.

Trellis Codes

Rather than use a convolutional code designed for maximum Hamming distance on a Euclidean-distance channel, one can design the code directly for maximum Euclidean distance. The Ungerboeck trellis codes are designed in this way for either real or complex Euclidean-distance channels. Figure 9 portrays a trellis for an Ungerboeck trellis code in contrast to uncoded QPSK, which is so popular for satellite communications; the code has the same bandwidth but requires 4.1 dB less signal-to-noise ratio. The trellis code replaces the signal constellation in the QPSK waveform with an 8-ary PSK signal constellation so as to introduce eight points in the complex plane that represent amplitude and phase modulation (or, in this example, phase modulation only). Instead of using the eight points to increase the data rate by modulating three bits at a time into a channel symbol, the trellis code modulates only two bits of data at a time into a three-bit channel symbol. The three code bits define one of eight possibilities that are labeled in the phase diagram on the right-hand side of Fig. 9. Every pair of incoming data bits is mapped into three code bits by the trellis encoder in order to create a waveform. This action is performed by shift-register circuits that are encoding data based on the memory of the two previous pairs of bits into the encoder, rather than on the current two bits alone.

The transmitted trellis-coded waveform has the same data rate as a QPSK waveform and the code is transparent to the user. The waveform has the same

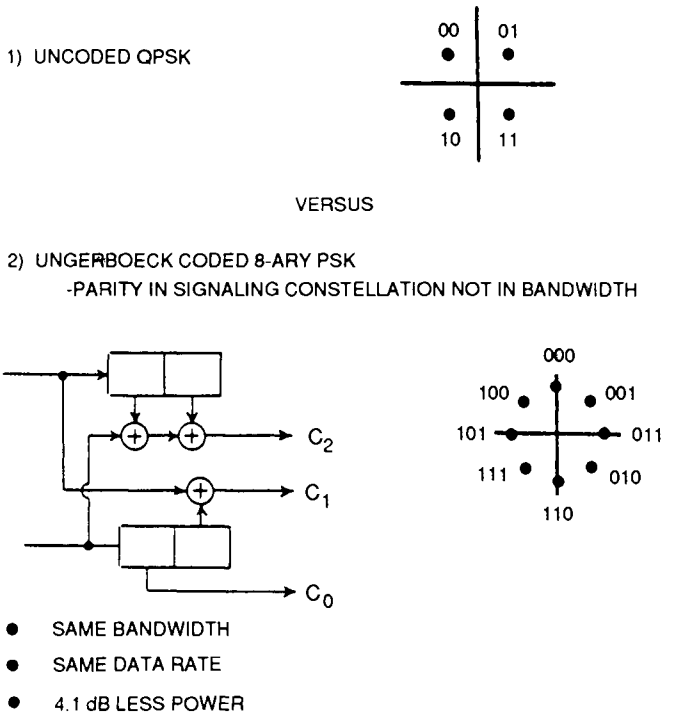

FIG. 9 Ungerboeck codes for reduction of signal-to-noise ratio.

bandwidth and the same data rate, but the power requirement is reduced by 4.1 dB. We are free to spend that 4.1 dB of now unneeded power in any desired fashion.

Ungerboeck codes can be used as a means of increasing the transmitted data rate at a fixed signal-to-noise ratio. Much more complicated signal constellations are required for this purpose. Signal constellations with as many as 256 points are now considered practical.

Matched Spectral-Null Codes

So far, we have only considered memoryless channels. Except for additive errors or noise, the channel output is the same as the channel input. In this section, we comment briefly on methods developed recently to construct codes that exploit the spectral characteristics of the channel.

Matched spectral-null codes combine the methods of trellis codes with the methods of partial-response signaling. The codewords, which are the input to the channel, are designed to obtain large minimum distance at the output of the channel. These codes exploit the memory in a channel to improve performance. Matched spectral-null codes are designed to exploit the observations that the minimum distance of a real-valued trellis code can be bounded by the order of the spectral null, and spectral nulls can be created in a sequence by constraining running digital sums.

The first-order running digital sum of a sequence $\mathbf{c}$ is defined as

$$s_\ell = \sum_{i=0}^{\ell} c_i$$

and the Kth-order running digital sum is defined recursively in terms of the $(K-1)$th-order running digital sum as

$$s_\ell^{(K)} = \sum_{i=0}^{\ell} s_i^{(K-1)}.$$

The Kth-order running digital sum can be understood as the output of the cascade of K accumulators.

For example, the sequence of coefficients

$$\mathbf{c} = (1,1,1,-7,2,2)$$

has a running digital sum given by

$$\mathbf{s}^{(1)} = (1,2,3,-4,-2,0).$$

The running digital sum of the running digital sum is

$$\mathbf{s}^{(2)} = (1,3,6,2,0,0)$$

The reason that the running digital sum is important is that the minimum Euclidean distance satisfies

$$d_{\min}^2 \geq 2K$$

if the Kth-order running digital sum is constrained to remain in a bounded interval. Moreover, if the channel itself has a spectral null of order L at zero frequency, then

$$d_{\min}^2 \geq 2(K + L).$$

By designing code sequences to satisfy the Kth-order running-digital-sum condition, coding gain can be realized.

This brief discussion suggests that unlimited coding gain is available simply by making K large. However, this is not so. Not only does the encoder become too complex, but other neglected effects become important and ultimately limit the achievable gain.

Synchronization Codes

A communication system that uses a block code (or a tree code) for data transmission will concatenate codewords (or code frames) into an indefinitely long string of codeword symbols. To decode an individual codeword, the decoder must be able to identify the boundaries between the codeword blocks (or frames). This requires a method to acquire block synchronization initially. Moreover, a channel may insert one or more false symbols into the datastream or delete one or more symbols from the datastream causing the decoder to misframe subsequent blocks. This requires a method to maintain or recover block synchronization.

A simple synchronization procedure is to insert a fixed sequence of symbols, known as a *marker*, into the codestream periodically, perhaps after every codeword. For example, the sequence 10111000 is an 8-bit binary marker that could be inserted after every codeword as an additional overhead. The codestream can be broken into codewords by recognizing the periodic occurrence of this pattern. However, the use of a marker in this way has two weaknesses. The marker is used outside of the error correction; in very noisy applications that use powerful error-prevention codes, there will be an unacceptable probability that all markers in a long interval will be in error. A second weakness is that nothing prevents a sequence of symbols of the codeword from forming a marker by chance. Indeed, this occasionally will happen if the data is random. More rarely, it can even happen in each of a long sequence of successive blocks. In a system transmitting millions of bits per second, even if the probability of a false marker is very low, it can still cause occasional unacceptable system interrupts.

Block codes can be constructed that are capable of recovering synchronization even in the presence of channel errors. If the system is initially synchronized

and thereafter no synchronization slippage of more than a few symbols will be encountered, then a self-synchronizing code may be employed to recover synchronization. Such codes can be designed to combine error correction and synchronization.

In the absence of errors, the preferred class of self-synchronizing codes are those for which the synchronization correction is not data dependent. This means that no concatenation of two codewords can form a codeword at the boundary. A *comma-free code* is a set of q-ary n-tuples such that for every pair of codewords $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ and $\mathbf{c}' = (c'_0, c'_1, \ldots, c'_n)$, the n-tuple $(c'_i, \ldots, c'_{n-1}, c_0, \ldots, c_{i-1})$ is not a codeword for any nonzero i. For example, the set

$$= \{10000,10001,10011,10111\}$$

is a binary (5,2) comma-free code. A concatenated stream of such codewords can always be synchronized, even if the start of the sequence is lost. Thus

$$\ldots 011001110001101111000010 \ldots$$

can only be punctuated as

$$\ldots 01,10011,10001,10111,10000,10 \ldots$$

Long, comma-free codes are more powerful than synchronization markers because there is no possibility of false synchronization, and more efficient than markers because comma-free codes with high rates exist. However, powerful methods for constructing such codes are not yet known nor has error correction been incorporated into such codes. Good constructive encoding and decoding algorithms for long, comma-free codes have not yet been developed. Encoding of a (100,92) comma-free code by a look-up table would not be practical.

A Reed-Solomon code is not a comma-free code because a comma-free code cannot have a codeword that is a cyclic shift of another codeword. However, a Reed-Solomon code can be modified to correct random errors and some synchronization errors simultaneously.

Suppose that a concatenated stream of codewords from a t-error-correcting Reed-Solomon code has a q-ary symbol deleted from the codestream. Then, all codewords that follow the codeword containing the deletion will be misframed. The first symbol of a received word will be the second symbol of a transmitted word, and so forth. The last symbol of the received word will be the first symbol of the next transmitted codeword. The last symbol will appear to be an error and, if there are at most $t-1$ other symbol errors, will be changed by the Reed-Solomon decoder to the missing first symbol of that codeword. Thus, the concatenated stream of corrected output words from the decoder will be correct, but each will appear cyclically shifted by one symbol position within its timeslot. Subsequent logic will eventually recognize that the last symbol of almost every codeword requires correction and will interpret this as a synchronization offset that then can be corrected in both future data and past data. In this way, a t-error-correcting Reed-Solomon code usually can correct ν random errors and

a synchronization slippage of $t - \nu$ symbols simultaneously. If the synchronization slippage is more than t symbols, the Reed-Solomon code cannot recover synchronization.

The Reed-Solomon code can be modified so that synchronization is not data dependent. A *coset code* of a code is the set $\{\mathbf{c} + \mathbf{h}\}$ where $\mathbf{c}$ is a codeword and $\mathbf{h}$ is a fixed vector. To encode into a coset code, first encode into the underlying Reed-Solomon code, then add the components of $\mathbf{h}$ to the components of the codeword. The sum forms the channel word. To decode, first subtract $\mathbf{h}$, then decode to a codeword. Adding $\mathbf{h}$ and then subtracting it has no effect on the ability to correct random errors. It does, however, ensure that synchronization slippages are correctable.

Specifically, choose $\mathbf{h}$ given by $h_0 = 1$ and $h_i = 0$ for $i \neq 0$. After a synchronization error of r symbols, the received word is partly one codeword and partly the following codeword and the 1 appears in the rth component. After the template is subtracted prior to the Reed-Solomon decoder, there will be two extraneous 1s, one in component 0 and one in the rth component, that are added to the Reed-Solomon codeword. These particular error patterns (two 1s separated by r symbols) occurring in every block are an indication of a synchronization error of r symbols.

Bibliography

Blahut, R. E., *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, MA, 1983.

Blahut, R. E., *Digital Transmission of Information*, Addison-Wesley, Reading, MA, 1990.

Clark, G. C., Jr., and Cain, J. B., *Error-Correction Coding for Digital Communications*, Plenum, New York, 1981.

Lin, S., and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

RICHARD E. BLAHUT

Coding of Facsimile Images

Introduction and Perspective

In digital facsimile systems we scan an image, convert the resulting analog signal to binary digital form, and then encode it to reduce the number of bits that represent the image. Advantages of coding include reduced transmission time and bandwidth requirements, with concomitant smaller transmission costs and speedier user service. Numerous facsimile coding studies have been made and various coding methods developed, each with particular advantages under specified conditions. In this article, we review the major approaches taken for coding of bilevel facsimile images and discuss issues relating to their performance, use, and future development.

We start with the initial capture of the source image, progressing through the processes of sensing, sampling, and quantization. Next we consider mathematical modeling of image generation, which provides an important tool in the analysis and design of coding methods. The models are based on information theory and the underlying theory of random or stochastic processes. The next section discusses the major approaches to *exact* or *lossless* coding. In this type of coding, the decoded image is exactly the same as the original digitized image. Included in this section is an overview of the recommendations for facsimile coding adopted by the International Telegraph and Telephone Consultative Committee (CCITT) and also a discussion of figures of merit for comparing coding methods. A coding process that is not exact is called an *approximate* or *lossy* one. The last section gives the major approaches to approximate coding and also discusses associated distortion issues.

Compression performance has often played the primary, if not sole, role in determining the relative worth of different coding methods. A host of other considerations are also of importance. Among these are design, production, and marketing concerns, including the implementation cost of both logic and storage, and the extent of design and testing effort. Of operational importance are speed of encoding and decoding, sensitivity to channel errors, and code synchronizability. Yet other figures of merit are the processability of the encoded data and the robustness and compatibility of the coding method. *Processability* is the ability to perform image operations directly on the encoded data. *Robustness* is the ability to handle image data variations without compression penalty. *Compatibility* is the ability of a method to convert into or evolve into methods that permit pattern-recognition functions, approximate codings, or other image operations, and the ability to work smoothly with other modules addressing wider functionality. For lossy encodings, an important figure of merit is perceived image quality.

The initial development of digital facsimile systems was based on several assumptions, including that the system should operate in a standalone mode, the system should handle only bilevel (black and white) images, and the received output should be displayed on hardcopy, usually paper. The underlying assump-

DOI: 10.1201/9781003209867-2

tion was that typewritten pages and drawings comprise the major use of facsimile. Based on these assumptions, the CCITT adopted a set of recommended facsimile coding methods. Most facsimile systems today meet one or more of these standards, and the resulting compatibility has encouraged substantial growth of the facsimile industry. However, system designers currently are addressing a number of issues not envisioned in the initial period of development.

Because of technological advances and new consumer demands, significant changes are underway in the uses and designs of facsimile systems. These changes are likely to have an impact on our view of coding processes. A particularly important trend is the migration of the facsimile function into computer systems. This currently takes on a variety of architectural forms, affecting where and how the facsimile coding methods are implemented. Along with the increasing role of computers, there is an emerging view of an integrated capability that embraces disparate communication modes, including electronic mail, voice mail, and facsimile, and requires the conversion from the format of any one mode to the format of any other (1). An allied trend is the use of soft facsimile, which is the display of received transmissions on erasable media, such as cathode-ray tubes (CRTs) and liquid-crystal display (LCD) panels, instead of paper. An important application of soft facsimile is browsing of remote large image databases. For this purpose, it is necessary to address coding needs for both storage and transmission, which may differ, as in the use of individual versus ensemble code books, for example (2). Also, browsing is likely to occur on low-capacity channels, where progressive transmission and display are necessary to make both informational and entertainment services attractive. In progressive modes, a crude representation of an image is transmitted first and finer detail is added in stages.

Still other facsimile-system trends include increased demand for higher resolutions, for color images, and also for increased multilevel capability. Some facsimile systems today offer digital halftone capability, using a bilevel representation of a multilevel image that gives the appearance of a gray-level image similar to pictures printed in newspapers. Current devices simulate from about 16 to 64 gray levels. In a soft facsimile environment, demand for 256 and more gray levels can be expected. Several color facsimile machines are now available. Additionally, there is a growing demand for composite documents, consisting of separately encoded scanned material, ASCII (U.S.A. Standard Code for Information Interchange) or other coded characters, graphics commands, and computer-generated images (3). Future scenarios may also include the extension of facsimile to provide such services as combined voice/picture mail and, more generally, multimedia messaging systems and other services that mix facsimile, still images, text, voice, and motion video. Such services are in step with advances in scanning and display technology and increased available bandwidth, such as that with broadband Integrated Services Digital Networks (ISDNs).

In response to increased capabilities and demands, many new facsimile features and architectures are already emerging. The traditional standalone facsimile unit is now supplemented with a variety of other ways to handle the facsimile function. These include numerous facsimile modem boards, alone or bundled with software packages that convert computer files into facsimile format and provide numerous other features. Some of these boards are equipped with a

small computer system interface (SCSI) for linking to scanners, printers, and mass storage units, and some boards have both data and facsimile modem capability. In another approach, the facsimile modems are external portable units that work from any personal computer (PC) serial port. Another configuration consists of a full facsimile machine that interfaces to a PC, local-area network (LAN) station, or mainframe host. It can operate in a standalone mode or as a scanner and backup printer for the computer system. Also of note are software packages for facsimile network servers equipped with facsimile modem boards that connect to standalone facsimile machines for scanning purposes and to computer printing devices for output. Offered more recently are products, referred to as *hydra* systems, that combine such multiple functions as scanning, facsimile, laser printing, digital voice recording, data modem communications, graphics display, and copying. In some cases, a hydra system also links to a computer. Also available are special chip products for the original equipment manufacturer (OEM) market, including a single chip that combines functions of modem signal processing, image compression, image enhancement, protocol processing, and control of mechanical parts.

In addition to the offerings above, there are also special software packages for image compression, some of which include the CCITT facsimile coding methods. These packages often are coupled with general graphics processors and with such peripherals as scanners, random-access storage devices, and streaming-tape drives. Raw data consists of both non-image and image types, and the image type includes bilevel, multilevel, and color images. Of particular interest, the coding methods generally used are more advanced than the standard CCITT facsimile methods, and they yield significantly higher compressions than do the CCITT methods. This further indicates that newer coding methods may prove desirable in an integrated systems approach that addresses both storage and transmission.

In addition to the basic facsimile functions, the innovations above include a rich offering of other features. Presentation abilities include monitor previewing of outgoing documents, monitor viewing of incoming documents with a choice of printing and/or computer storage, the ability to rotate images to correct for upside-down scanning, and to zoom at one-third, one-half, and full size. Other user conveniences are preformatted cover sheets (including letterhead, logo, and signature), phone directory services, background facsimile reception, and automatic notification of incoming messages. Conversion features include converting ASCII text to facsimile images and converting facsimile images to ASCII text, which enables editing of received messages. Among networking abilities are sending composite facsimile, text, and binary files in one transmission, sending in a broadcast mode to multiple destinations, delayed sending to take advantage of off-hours, unattended sending and receiving, automatic dialing and automatic redialing in case of a busy destination or an interruption, automatic time and date stamping, automatic logging of facsimile documents sent and received, store-and-forward capability for facsimile traffic from other terminals, and automatic extraction of routing instructions from incoming documents and transfer to indicated destinations.

In connection with this diversity of options, there is a need for continued systems analysis to integrate different solutions and to provide for graceful

growth of this industry, including reassessment of facsimile coding needs and the need for expanded standards that incorporate new functional parameters and appropriate coding methods. This article reviews some of the options available for bilevel facsimile coding; additional sources of information are listed in the Bibliography.

Capturing the Source

Introduction

The facsimile coding process consists of separate encoding and decoding phases. These are part of a sequence of functions performed in a digital facsimile system, which are shown in a generic form in the diagram in Fig. 1. In traditional standalone facsimile systems, the functions shown generally are incorporated in a single unit. As discussed above, in newer architectures, these functions are distributed in various ways among computers, their peripherals, and other units, including self-contained facsimile systems.

In Fig. 1, the first facsimile function is the scanning of a source image. Scanning consists of three phases. First, the image is sensed, obtaining an electrical analog signal that varies in accordance with the optical intensity variations of the image. Then, the analog signal is sampled, producing a discrete set of real-valued sample values of the analog signal. Finally, these samples are quantized, resulting in a discrete sequence of digital values consisting of equal-length binary words. These three processes are discussed in separate sections below, particularly as they refer to bilevel (two-tone, usually black and white) and multilevel (monochrome or gray-level) images.

Sensing the Source Image

To sense an image, it is first illuminated by light. For paper images, this light is reflected from the medium; for film images, it passes through the medium. The reflected or transmitted light varies in intensity according to the image content. In either case, a photoelectric device converts the light variations into an analog electrical signal. There are a variety of devices that perform illumination. They vary in the portion of the image that is illuminated by the source light at any given time, in the way the source light is delivered to the image, in the type of source light, and in the way of sensing the reflected or transmitted light.

In a common type of device, the image is illuminated and sensed in strips or lines across the width of the image, one strip at a time, from the top to the

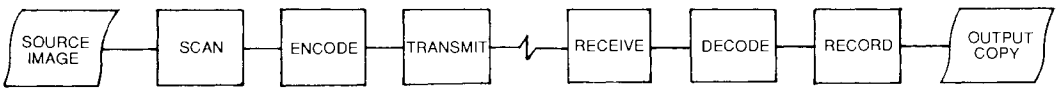

FIG. 1 Digital facsimile system functions.

bottom of the image (see Fig. 2-*a*). In one arrangement, the light source is a fluorescent tube extending across the strip being sensed (4). The reflected light is focused through a lens system onto a smaller field containing a linear array of photodiodes, charge-coupled devices (CCDs), or charge-injection devices (CIDs), which convert light into electrical form. In many of these devices, the image lies on a flat bed and remains stationary while the light source and sensors are moved to access successive image strips. In another arrangement, light from an incandescent source travels to the image through a set of optical fibers fitted closely to the image strip being sensed (5). A second set of optical fibers, also fitted closely to the image strip being sensed, pick up the reflected light and transmit it to a CCD device, which again provides the conversion to electrical form. In this arrangement, the imaging components are stationary and the image steps through a rollfeed mechanism to access successive image strips.

Lasers or light-emitting diodes (LEDs) also can supply the source light. Or, the scanning device may be a video camera that focuses an entire optical image onto a plate of photosensitive material, which acquires an electrical pattern varying in accordance with the optical image. An electron beam then scans the plate in a line-by-line fashion, causing the pattern to be converted into signal form. Also available are two-dimensional arrays of CCDs, onto which an entire image is focused. Yet another approach illuminates and senses a single small

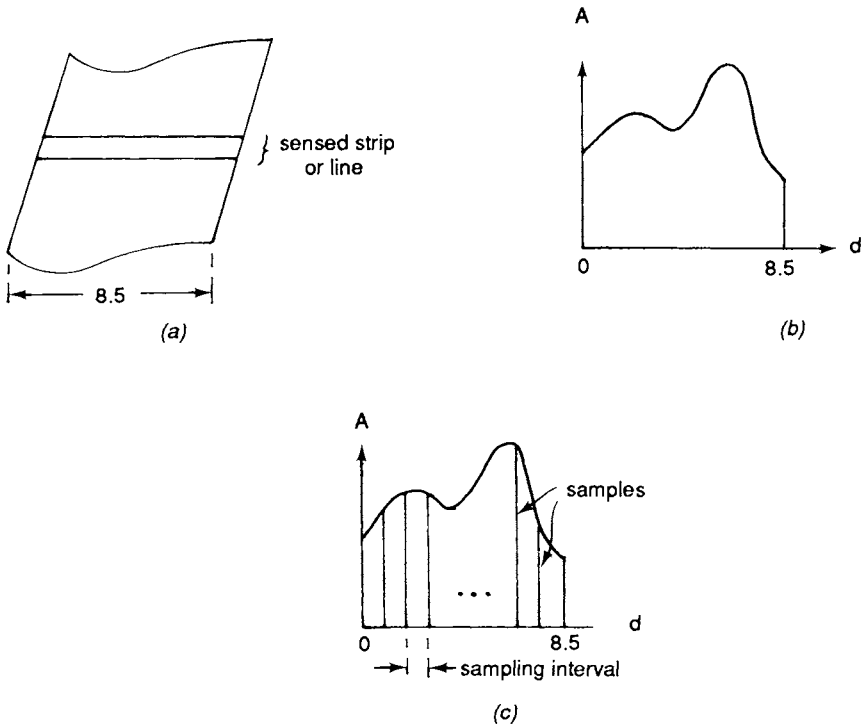

FIG. 2 Sensing and sampling the source image (A = amplitude; d = distance from left edge or time of sensing): *a*, document being sensed; *b*, sensed signal for one image line; *c*, sampled signal for one image line.

spot at a time. A version of this is the rotating-drum scanner, often used in graphic-arts applications. The image is wrapped around the drum, and is illuminated by a spotlight. The light source and the sensor mechanism are contained in a housing mounted on a carriage that permits linear travel parallel to the drum axis. Scanning is provided in the X direction by the drum rotation and in the Y direction by the linear movement of the sensor. Flying-spot scanners are another version of illuminating and sensing a small spot at a time. Other arrangements are also possible.

In most scanning devices, the output signal is nonuniform over the image area. This is called *shading*, and can be due to uneven lighting, lens curvature, electron-beam positioning error, or variations in photosensor sensitivity. Correction circuitry compensates for these aberrations. Some newer devices also provide for calibration of each particular device the first time it is used; direct current (DC) offset voltage and individual photosite responses are stored in separate tables and used dynamically to correct data values as they are acquired (6).

Sampling the Analog Signal

The result of the sensing operation is an analog electrical signal (see Fig. 2-*b*). To convert this signal to digital form, it is first sampled, and the resulting samples are then quantized. *Sampling* is the process of obtaining signal amplitude values taken at successive and, usually, evenly spaced points along the distance or time coordinate of the analog signal (Fig. 2-*c*). The *sampling interval* is the distance between two successive points on the coordinate, and its inverse is the *sampling rate*.

The question arises as to what sampling rate should be used. In theory, if the highest frequency contained in the signal is W (cycles per unit of distance or time), then the sampling rate should be at least $2W$. This follows from the classical sampling theorem proposed by Nyquist (7), which states that it is possible to reconstruct completely a signal from its samples if the sampling rate is equal to twice the signal bandwidth, $2W$. This is called the *Nyquist rate*. The theorem also holds if the sampling rate is greater than $2W$. If a sampling rate less than $2W$ is used, a condition called *aliasing* occurs, wherein the spectrum of the sampled signal contains overlaps of adjacent spectral segments in the periodic spectrum. Figure 3 illustrates the effect of sampling rate on an ideally sampled one-dimensional signal. The spectrum of the signal to be sampled is shown in Fig. 3-*a*. The spectrum of the ideally sampled signal using a sampling rate greater than $2W$ (Fig. 3-*b*) is reconstructed exactly by an ideal low-pass filter of range W. This also applies where the sampling rate exactly equals $2W$ (Fig. 3-*c*). However, when the sampling rate is less than $2W$, adjacent spectral segments overlap, resulting in the spectrum indicated by the solid line at the top of Fig. 3-*d*. Applying an ideal low-pass filter of range W to this spectrum yields the spectrum illustrated in Fig. 3-*e*. This clearly is not the spectrum of the original signal depicted by Fig. 3-*a*; the altered spectrum of Fig. 3-*d* indicates the aliasing error. Figure 3 illustrates aliasing for a one-dimensional signal. A

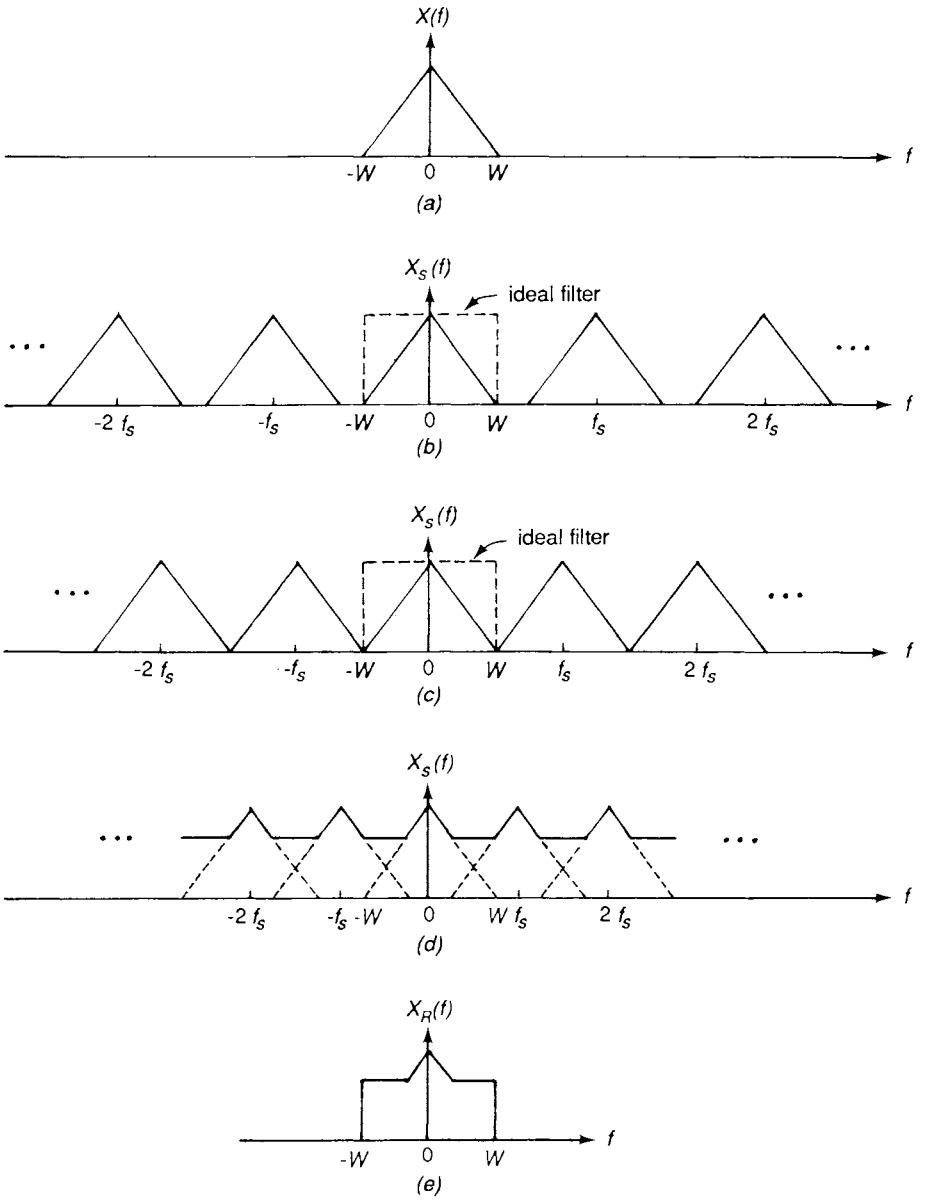

FIG. 3 Effect of sampling rate: *a*, spectrum of signal; *b*, spectrum of ideally sampled signal at sampling rate of $f_s > 2W$; *c*, spectrum of ideally sampled signal at sampling rate of $f_s = 2W$; *d*, spectrum of ideally sampled signal at sampling rate of $f_s < 2W$; *e*, spectrum of ideally reconstructed signal at sampling rate of $f_s < 2W$, showing aliasing.

similar analysis applies for an image signal defined in two dimensions, where aliasing results in moire patterns in the reconstructed image (8).

Aliasing can be reduced by increasing the sampling rate, which also increases the number of bits needed to represent an image. An alternative is to low-pass filter the analog image signal before sampling. Filtering is mathematically equivalent to finding the convolution of the image signal with a filter function. A common filter function has a value of 1 over a unit square, a value of 0 elsewhere, and effectively performs local neighborhood averaging throughout an image. Area averaging is effective along polygon edges, but is less effective on lines that are thinner than the sampling interval; Gaussian filters give better total performance (9). Averaging also can be accomplished in optical spot scanners by designing the filtering function into the lens system or by slightly defocusing the spot beam.

In practice, sampling rates suitable for scanning images have been determined by subjective criteria. The needs differ depending on the type of image and its intended use. The results of several subjective studies on bilevel material follow. For typewritten pages, 100 samples, or dots per inch (dpi), achieve legibility, but 150 dpi are required for acceptability (10). This increases to 200 dpi for Chinese characters (11). For weather maps, the requirement is 100–200 dpi, and for fingerprints it is 200–400 dpi (12). For engineering drawings, 200–250 dpi is used (13,14). Text in newspaper pages requires 400 dpi for satisfactory results (15). Material of graphic-arts quality, such as that found in the White Pages and Yellow Pages of telephone directories, requires from about 400 dpi to 2000 dpi (16,17).

Quantizing the Samples

The result of the sampling operation is a set of real-valued samples of the analog signal such as those illustrated by Fig. 2-*c*. The next facsimile function to be performed is quantization. In this discussion, we address *scalar quantization*, which maps each real-valued sample separately into one of a finite set of quantized values. An extension of this is *vector quantization*, which maps a vector of real-valued samples (corresponding to a block of image picture elements) into one of a finite set of vectors of quantized values. *Binary-vector quantization* is a form that maps a vector of quantized values into one of a smaller set of vectors of quantized values; this is discussed below in the section concerning coding.

For scalar quantization, the possible range of sample values is divided into a discrete set of ranges called *quantization intervals*. Each interval is assigned a unique symbol from a symbol alphabet that usually consists of a set of equal-length binary words. Quantization is the process of mapping each sample value into the symbol of the interval containing the value. For a bilevel quantization, there are only two alphabet symbols, 0 and 1. An example of bilevel quantization intervals and symbols is shown in Fig. 4-*a*.

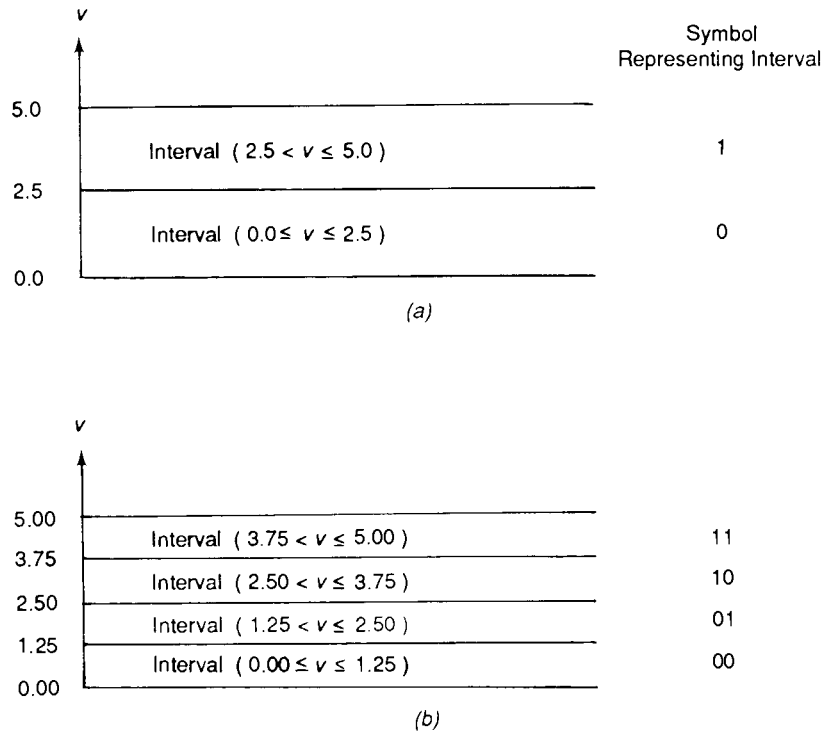

FIG. 4 Quantization intervals and symbols (v = signal amplitude or sample value): *a*, bilevel quantization; *b*, multilevel quantization.

The boundaries of the intervals are called threshold values or decision levels because they are used to decide the interval to which a sample belongs. It is not necessary or even optimal for the two intervals to be of the same size. For typewritten characters, the position of the decision level between the two intervals is critical. Setting it too far in one direction causes the reproduced type to appear bloated, and setting it too far in the other direction causes it to appear spindly. The latter case results in fewer bits required to code the image. In making comparisons of image coding methods, it is important that the same digitized material be used for each method, otherwise an apparent advantage of one method may be due to a favorable decision level.

When more than two quantization intervals are used, the process is called *multilevel* or *gray-level quantization*. Figure 4-*b* offers an example of multilevel quantization that defines four intervals, represented by the symbols 00, 01, 10, and 11.

The set of symbols or digital values resulting from the quantization of an image are collectively referred to as the image *bit map*. Each symbol corresponds to a subarea of the image termed a *picture element*, contracted to *pel* or *pixel*. Figure 5 presents excerpts of bit maps arranged in matrix form corresponding to the pels of the original image; Fig. 5-*a* is an example of a bilevel image and Fig. 5-*b* illustrates a multilevel image digitized to eight levels.