INFORMATION THEORY OF DATA TRANSMISSION CODES

The basic task of a communication system is to extract relevant information from a source, transport the information through a channel and to reproduce it at a receiver. Shannon, in his ground-breaking *A Mathematical Theory of Communications* (1), quantified the notions of the information rate of a source and the capacity of a channel. He demonstrated the highly non-intuitive result that the fundamental restrictive effect of noise in the channel is not on the *quality* of the information, but only on the *rate* at which information can be transmitted with perfect quality. Shannon considered *coding* schemes, which are mappings from source outputs to transmission sequences. His random-coding arguments established the existence of excellent codes that held the promise of nearly zero error rates over noisy channels while transmitting data at rates close to the *channel capacity*. Shannon's existence proofs did not, however, provide any guidelines toward actually constructing any of these excellent codes. A major focus of research in *information theory* (as Shannon's theory came to be known) over the past 50 years following Shannon's seminal work has been on constructive methods for channel coding. A number of later books (e.g., Refs. 2–9) journalize the development of information theory and coding. In this article we present a broad overview of the state of the art in such data transmission codes.

DATA SOURCES AND CHANNELS

A very general schematic representation of a communication link consists of the following cascade: the source, a source encoder, a channel encoder, a modulator, the channel, the demodulator, the channel decoder, the source decoder, and the receiver. The *source encoder* typically converts the source information into an appropriate format taking into account the quality or *fidelity* of information required at the receiver. Sampling, quantization and analog-to-digital conversion of an analog source, followed possibly by coding for redundancy removal and data compression, is an example. The codes used here are usually referred to as *data compaction* and *data compression* codes. Both strive to minimize the number of bits transmitted per unit of time, the former without loss of fidelity and the latter with possible, controlled reduction in fidelity. This source encoder is followed by the *channel encoder,* which uses *data transmission codes* to control the detrimental effects of channel noise. Controlled amounts of redundancy is introduced into the data stream in a manner that affords error correction. These data transmission codes are the focus of this article. Further down in the cascade, we have the *modulator* which maps output strings from the channel encoder into waveforms that are appropriate for the channel. (Traditionally, modulation has evolved as an art disjoint from coding, but some recent research has indicated the merits of combined coding and modulation. We will touch upon this aspect toward the end of this article.) Following the channel, the demodulator and the decoders have the corresponding inverse functions which finally render the desired information to the receiver. In brief, this article concentrates on data transmission codes.

Binary Symmetric Channels

Each binary digit or (group of digits) at the input of the modulator is transmitted as a waveform signal over the transmission channel. Physical transmission channels may distort the signal, the net result of which is to occasionally reproduce at the output of the demodulator a binary string that is different from what was actually sent. In many practical cases, the error events in successive binary digit positions are mutually statistically independent. And in many such binary memoryless channels the probability of error, ϵ, is the same for a transmitted 0 as well as for a transmitted 1 (Fig. 1). Such a *binary symmetric channel* (BSC) is an important abstraction in data transmission coding.

If a binary n-vector is transmitted sequentially (i.e., bit by bit) over a binary symmetric channel with bit error probability ϵ, the number of errors is a random variable with a Bernoulli distribution:

$$P(i) = \binom{n}{i} \epsilon^i (1-\epsilon)^{n-1}, \qquad 0 \le i \le n$$

If $\epsilon < \frac{1}{2}$, as is the case for most practically useful channels, $P(i)$ is seen to diminish exponentially in i as $(\epsilon/1 - \epsilon)^i$. This implies that $P(0) > P(1) > P(2) > \cdots > P(n)$. More specifically, $P(0)$ is typically very large, $P(1)$ is $O(\epsilon)$ (i.e., on the order of ϵ), $P(2)$ is $O(\epsilon^2)$, and so forth. Thus, even minimal levels of error correction can bring about a significant performance improvement on the BSC.

Hamming Distance and Hamming Weight

The BSC can be modeled as a mod-2 additive noise channel characterized by the relation $Y = X \oplus N$, where X is the

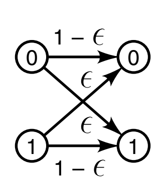

Figure 1. The binary symmetric channel with error probability ϵ.

transmitted binary digit, N is a noise bit, Y is the corresponding output bit, and "$\oplus$" denotes mod-2 addition. The *Hamming weight* of a binary n-vector is defined as the number of 1's in it, and the *Hamming distance* [in honor of R. W. Hamming, a coding theory pioneer (10)] between two binary vectors is defined as the number of bit positions where the elements of the two vectors are different. It is easy to see that the mod-2 sum of two binary n-vectors has a Hamming weight equal to the Hamming distance between the two vectors. If a binary input n-vector $\boldsymbol{X}^n$ to a BSC produces the output n-vector $\boldsymbol{Y}^n$, then the noise pattern $\boldsymbol{N}^n = \boldsymbol{X}^n \oplus \boldsymbol{Y}^n$ is a binary n-vector whose Hamming weight is equal to the Hamming distance between $\boldsymbol{X}^n$ and $\boldsymbol{Y}^n$. (If $a \oplus b = c$, then $a = b \oplus c$).

Consider the n-space $\boldsymbol{S}^n$ of all binary n-vectors. Out of the total 2^n n-vectors in $\boldsymbol{S}^n$, if we choose only a few vectors well separated from each other, we can hope that noise-corrupted versions of one codeword will not be confused with another valid codeword. To illustrate, suppose we choose a code with two binary n-vector codewords $\boldsymbol{X}_1^n$ and $\boldsymbol{X}_2^n$ which are mutually at a Hamming distance d. In Fig. 2 we have shown the example of $\boldsymbol{S}^4$ with $\boldsymbol{X}_1^4 = (0000)$ and $\boldsymbol{X}_2^4 = (1111)$ at Hamming distance $d = 4$. ($\boldsymbol{S}^4$ is a four-dimensional hypercube, with each node having four neighbors at unit distance along the four orthogonal axes.) It can be seen that if codeword 0000 has at most one bit altered by the channel, the resulting 4-tuple (e.g., 0001) is still closer to 0000 than to 1111 so that a nearest-codeword decoding rule decodes correctly. But if 0000 encounters two bit errors (e.g., 0011), the resulting word is at equal distance from either codeword; and if there are three bit errors (e.g., 1101), the nearest codeword now is 1111 and a decoding error results. In general, two codewords at a Hamming distance d can be correctly decoded if the number of errors incurred in the BSC is at most $\lfloor (d-1)/2 \rfloor$, where $\lfloor x \rfloor$ is the integer part of the number x. If in fact there are more than two codewords in the code, it should be obvious that the pair of codewords with the minimum Hamming distance determine the maximum number of bit errors tolerated. Thus, a code with minimum distance $d_{\min}$ can correct all error patterns of Hamming weight not exceeding $t = \lfloor (d_{\min} - 1)/2 \rfloor$.

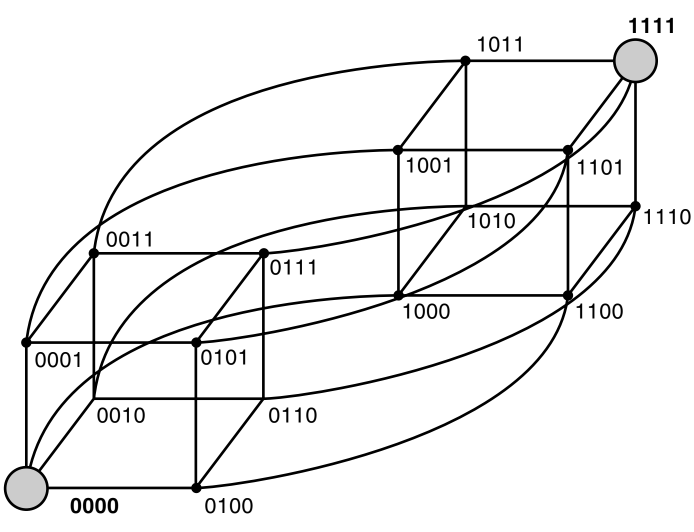

Figure 2. Minimum distance decoding. The two codewords 0000 and 1111 are at Hamming distance 4 in the space of binary 4-tuples.

The essence of code design is the selection of a sufficient number of n-vectors sufficiently spaced apart in binary n-space. Decoding can in principle be done by table lookup but is not feasible in practice as the code size grows. Thus we are motivated to look for easily implemented decoders. Such practical coding schemes fall generally into two broad categories: block coding and convolutional coding.

BLOCK CODING

Linear Block Codes

An (n, k) *block code* maps every k-bit data sequence into a corresponding n-bit codeword, $k < n$. There are 2^k distinct n-vector codewords in a linear block code. The *code rate* $R = k/n$ is a measure of the data efficiency of the code. A linear block code has the property that for any two codewords $\boldsymbol{X}_i^n$ and $\boldsymbol{X}_j^n$, their bitwise mod-2 sum $\boldsymbol{X}_i^n \oplus \boldsymbol{X}_j^n$ is also a codeword. Using a geometric perspective, we can view the code as a k-dimensional linear subspace of the n-dimensional vector space $\boldsymbol{S}^n$, spanned by k basis vectors. Using matrix notation, we can then represent the linear encoding operation as $\boldsymbol{Y}^n = \boldsymbol{X}^k\boldsymbol{G}$, where the k-vector $\boldsymbol{X}^k$ is the data vector, $\boldsymbol{Y}^n$ is the corresponding n-vector codeword, and $\boldsymbol{G}$ is the $k \times n$ binary-valued *generator matrix.* The rows of $\boldsymbol{G}$ are a set of basis vectors for the k-space and thus are mutually linearly independent. Linear codes have the important feature that the minimum distance of the code is equal to the smallest among the nonzero Hamming weights of the codewords. (The all-zero n-vector is necessarily a codeword in each linear n-vector code.) If the codewords are of the specific concatenated form $\boldsymbol{Y}^n = (\boldsymbol{X}^k\boldsymbol{P}^{n-k})$, where $\boldsymbol{P}^{n-k}$ is a *parity vector* comprising $n - k$ *parity bits* which are solely functions of $\boldsymbol{X}^k$ (i.e., if the codeword $\boldsymbol{Y}^n$ contains the data word $\boldsymbol{X}^k$ explicitly), then the code is termed *systematic.* Systematic linear block codes have generator matrices with the special structural form $\boldsymbol{G} = [\boldsymbol{I}_k\boldsymbol{P}]$, where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix and $\boldsymbol{P}$ is a $k \times n - k$ parity generator matrix. Any linear block code can be put into an equivalent code that is also systematic. A (7,4) Hamming code (discussed below) is one example of a linear block code with the following generator matrix in its systematic form:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For every linear (n, k) block code, there is a *parity check matrix* $\boldsymbol{H}$ which is an $(n - k \times n)$ binary valued matrix with the property that $\boldsymbol{G}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0}$. Given $\boldsymbol{G} = [\boldsymbol{I}_k\boldsymbol{P}]$, the corresponding parity check matrix has the structure $\boldsymbol{H} = [\boldsymbol{P}^{\mathrm{T}}\boldsymbol{I}_{n-k}]$. The parity check matrix for the (7,4) systematic Hamming code is as follows:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The condition $\boldsymbol{G}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0}$ implies that every row in $\boldsymbol{G}$, and consequently every codeword, is orthogonal to every row in $\boldsymbol{H}$. Every codeword $\boldsymbol{X}^n$ satisfies the parity check condition $\boldsymbol{X}^n\boldsymbol{H}^{\mathrm{T}}$

$= \mathbf{0}$. For an arbitrary $\boldsymbol{Y}^n$ appearing at the output of a BSC, the $n - k$ vector $S(\boldsymbol{Y}^n) = \boldsymbol{Y}^n\boldsymbol{H}^T$ is called the *syndrome* of $\boldsymbol{Y}^n$.

The 2^{n-k} syndromes have a one-to-one correspondence with a set of 2^{n-k} n-vector error patterns that the (n, k) linear code is capable of correcting. If n is small, a table lookup will suffice to find the error pattern from the syndrome. A *standard array* (11) as shown below helps to mechanize this procedure:

$$\begin{bmatrix} 0 & X_1 & X_2 & \cdots & X_i & \cdots & X_{2^k-1} \\ N_1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ N_2 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ N_j & \cdots & \cdots & \cdots & Y^n & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ N_{2^{n-k}-1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

The top row of the standard array consists of the 2^k codewords. The first element, $\boldsymbol{N}_1$ in the next row is chosen to be an n-vector error pattern that the code is expected to correct. It must not be one of the elements in the preceding row(s). The succeeding elements of this row are obtained by adding this error pattern to the corresponding codeword in the top row. Additional rows are formed by repeating this procedure, each time choosing the first element of the row to be a pattern that has not appeared already in the rows above. Each row of the resulting $2^{n-k} \times 2^k$ standard array is called a *coset,* and the first element in each row a *coset leader.* For a BSC with error probability $\epsilon < \frac{1}{2}$, it is natural to choose the coset leaders N to have the least Hamming weight possible. Given the standard array, the output of a BSC, $\boldsymbol{Y}^n$, is located in the standard array. The codeword $\boldsymbol{X}_i$ at the top of the column that $\boldsymbol{Y}^n$ belongs to is declared as the transmitted codeword, with the error pattern produced by the BSC being the coset leader $\boldsymbol{N}_j$ for the coset that $\boldsymbol{Y}^n$ belongs to. If the BSC produces an error pattern which is not one of the coset leaders, the decoder will clearly make a decoding error. In the standard array for the (7,4) Hamming code, the coset leaders can be chosen to be the set of all 7-bit patterns with Hamming weight 1. Hence this code corrects all single error patterns and none else.

The matrix $\boldsymbol{H}^T$ generates an $(n, n - k)$ code (comprising all linear combinations of its $n - k$ linearly independent rows). The codes generated by $\boldsymbol{G}$ and $\boldsymbol{H}^T$ are referred to as dual codes of each other. The weight spectrum of a block code of length n is defined as the $(n + 1)$-vector $(A_0, \ldots, A_n)$, where A_i is the number of codewords with Hamming weight i. The MacWilliams identities (12) link the weight spectra of dual codes. In particular, if $k \gg n - k$, the weight spectrum of the (n, k) code with 2^k codewords may be obtained more easily from the weight spectrum of the dual code with only 2^{n-k} codewords, by means of the MacWilliams identities. The weight spectrum of a linear block code determines the probability of undetected error when the code is used over a BSC. Whenever the n-vector error pattern generated by the BSC coincides with one of the codewords, the error becomes undetectable by the code. This undetected-error probability is

$$P_{\text{UDE}} = \sum_{i=0}^{n} A_i \epsilon^i (1-\epsilon)^{n-i}$$

and is clearly an important performance parameter, especially when codes are used for error detection only.

For the general class of linear block codes, the encoder implements the multiplication of the data vector by the generator matrix. Decoding consists of computing the syndrome (by matrix multiplication) and looking up the corresponding coset leader in the standard array. These lookup procedures become difficult for codes with moderate to large values of block length n. This motivates the study of a subclass of linear block codes, namely, cyclic codes, with features that facilitate more easily implemented decoders.

Cyclic Codes

A cyclic code is a linear block code with the special property that every cyclic shift of a codeword is also a codeword. Cyclic codes were first proposed by Prange (13). Polynomial algebra, where binary vectors are represented by polynomials with binary coefficients, is a useful framework for characterization of cyclic codes. A binary n-vector $\boldsymbol{X}^n = (x_1, x_2, \ldots, x_n)$ has the polynomial representation $X(D) = x_1D^{n-1} + x_2D^{n-2} + \cdots + x_n$, with degree not exceeding $n - 1$. For instance, (0101) corresponds to $X(D) = D^2 + 1$. Analogous to the generator matrix of a linear block code, a cyclic code can be characterized in terms of a *generator polynomial* $G(D)$ such that every codeword has a polynomial representation of the form $X(D) = G(D)R(D)$. Here $G(D)$ is a polynomial of degree $n - k$ and $G(D)$ is a factor of $D^n - 1$. The polynomial $R(D)$ has degree $k - 1$ or less, representing the k-bit data vector $(r_1, \ldots, r_k)$ being encoded. A code polynomial is generated by multiplying the data polynomial $R(D)$ by the generator polynomial $G(D)$. It can be verified that multiplication of polynomials corresponds to convolution of the corresponding vectors. This observation leads to simple implementation of encoders using shift-register based digital circuits.

Denoting $G(D) = g_0 + g_1D + g_2D^2 + \cdots + g_{n-k}D^{n-k}$, the generator *matrix* $\boldsymbol{G}$ for the linear block code generated by $G(D)$ has the following form:

$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & . & . \\ . & . & . & \cdots & . & . \\ . & . & . & \cdots & g_{n-k} & 0 \\ 0 & 0 & 0 & \cdots & g_{n-k-1} & g_{n-k} \end{bmatrix}$$

As an example, binary cyclic codes of length $n = 7$ are generated by the factors of $D^7 - 1 = (D + 1)(D^3 + D + 1)(D^3 + D^2 + 1)$. The first degree polynomial $G_1(D) = D + 1$ generates the (7, 6) code with a single overall parity bit, while $G_2(D) = D^3 + D + 1$ results in the (7, 4) Hamming code.

The polynomial $H(D)$ such that $G(D)H(D) = D^n - 1$ is known as the *parity check polynomial* for the code generated by $G(D)$. [Since $H(D)$ is also a factor of $D^n - 1$, it also can generate a cyclic code, which is the dual of the code generated by $G(D)$.] The polynomial $H(D) = h_0 + h_1D + h_2D^2 + \cdots + h_kD^k$ specifies the form of the parity check matrix $\boldsymbol{H}$ of the code as follows:

$$H = \begin{bmatrix} h_k & h_{k-1} & \cdots & 0 \\ 0 & h_k & \cdots & h_0 \\ 0 & 0 & \cdots & h_1 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & h_k \end{bmatrix}$$

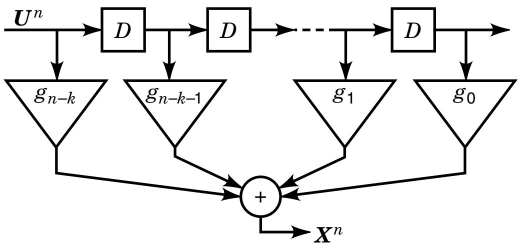

Figure 3. A generic cyclic encoder.

The special structure of $\boldsymbol{G}$ and $\boldsymbol{H}$ for cyclic codes greatly simplifies the implementation of the encoder and the syndrome computer. A generic encoder for a cyclic code is shown in Fig. 3. The k-bit data vector $\boldsymbol{U}^k$ is pipelined through the shift register for n clock times, thus generating the codeword $\boldsymbol{X}^n$ at the output. The encoder utilizes $n - k$ single-bit delay units D, binary multipliers, and binary adders. The circuit complexity is seen to grow only linearly in block length n. For decoding, we can develop a circuit for syndrome calculation for a received n-vector, structured very similarly to Fig. 3. Logic circuits are used to expand the $n - k$ bit syndrome into an n-bit error pattern which is then added to the received codeword to effect error correction.

BCH Codes

Bose and Ray-Chaudhuri (14) and independently Hoquenghem (15) discovered a remarkably powerful subclass of cyclic codes, referred to as BCH codes. The BCH code family is practically the most powerful class of linear codes, especially for small to moderate block lengths. BCH codes can be designed for a guaranteed *design distance* δ (which of course cannot exceed the true minimum distance $d_{\min}$ of the resulting code). Specifically, given δ and hence $t = \lfloor(\delta - 1)/2\rfloor$, and for any integer m, there is a t-error correcting binary BCH code with block length $n = 2^m - 1$ for which the number of parity check bits is no more than mt. Powerful algorithms exist for decoding BCH codes. The polynomial algebraic approach to BCH decoding was pioneered by Peterson (16) for binary BCH codes and extended to nonbinary BCH codes by Gorenstein and Zierler (17). Major contributions came later from Chien (18), Berlekamp (19), and Massey (20). An alternative approach to BCH decoding based on finite field Fourier transforms has gained attention recently, from the work of Blahut (21).

Reed–Solomon Codes

Reed–Solomon codes (22,23) are a very powerful generalization of BCH codes. Binary Reed–Solomon codes can be defined, for any integer m, as follows. Serial data is organized into m-bit symbols. Each symbol can take one of $n = 2^m$ values. The Reed–Solomon code block length is $N = 2^m - 1$ symbols, or Nm bits. Out of these, K symbols are data symbols (i.e., $k = mK$), and $N - K$ symbols are parity symbols computed according to the algebraic description of the code. Reed–Solomon decoding can recover from up to $t = \lfloor(N - K)/2\rfloor$ symbol errors. If symbol erasures are marked as such (i.e., if additional side information is available as to whether a symbol is in error or not, though it is not known what the errors are), then the Reed–Solomon erasure correction limit is $t = N - K$ symbol errors. Since the code can in particular correct t consecutive symbol errors or erasures, it is especially effective against burst errors. The Reed–Solomon codes are *maximum-distance separable;* that is, for the admissible choices of n and k, the Reed–Solomon codewords are spaced apart at the maximum possible Hamming distance.

Perfect Codes

An (n, k) linear code can correct 2^{n-k} error patterns. For some integer t, if the set of error patterns consists of exactly all error patterns of Hamming weight t or less and no other error patterns at all, such a code is termed as a perfect t-error correcting code. This would require, for binary codes, that n, k, and t satisfy the following equality:

$$\sum_{i=0}^{t} \binom{n}{i} = 2^{n-k}$$

The only known perfect codes are the Hamming codes, the double-error-correcting ternary Golay code, and the triple-error-correcting binary Golay code, described below. Tietvainen (24) proved that no other perfect codes exist.

Hamming Codes

Hamming codes are single error correcting codes. For $t = 1$, the condition for a perfect code becomes $1 + n = 2^m$ where $m = n - k$. For integers m, Hamming single-error-correcting codes exist with m parity check bits and block length $n = 2^m - 1$. The rate of the $(2^m - 1, 2^m - m - 1)$ code is $R = (2^m - m - 1)/(2^m - 1)$, which approaches 1 as m increases. The generator matrix G that was displayed earlier while describing linear codes is indeed a generator matrix for a Hamming (7, 4) code. It is possible to rearrange the generator matrix of the code so that the decimal value of the m-bit syndrome word indicates the position of the (single) errored bit in the codeword. Adding an overall parity bit to a basic Hamming code results in a $(2^m, 2^m - m - 1)$ code capable of detecting double errors in addition to correcting single errors. Such codes are particularly effective in data transmission with automatic repeat request (ARQ). If the bit error rate is $\epsilon < \frac{1}{2}$, then the single error patterns which appear with probability $O(\epsilon)$ are the most common and are corrected by the code, thus avoiding the need for retransmission. The double error patterns have a lower probability $O(\epsilon^2)$ and are flagged for retransmission requests. Other error patterns occur with negligibly small probabilities $O(\epsilon^3)$ or less.

Hamming codes are cyclic codes. For block lengths $n = 2^m - 1$, their generator polynomials are the primitive polynomials of degree m over the binary field. (An mth degree primitive polynomial with binary coefficients has the property that its m roots can be characterized as the m primitive elements in a finite field of 2^m elements.) In fact, Hamming codes are BCH codes as well. However, they are decodable by far simpler methods than the general BCH decoding algorithms. Most notably, Hamming codes are perfect codes.

Golay Codes

Two codes discovered by Golay (25) are the only other perfect codes, apart from the Hamming codes mentioned above. For $n = 23$ and $t = 3$, the total number of 0-, 1-, 2-, and 3-error binary patters of length 23 add up to $2048 = 2^{11}$. Choosing $m = 11$, we can form the (23, 12) triple-error-correcting Golay code. It is also possible to form an (11, 6) double-error-correcting perfect code over ternary alphabet. The Golay codes also are BCH codes, and hence cyclic codes.

Extended and Shortened Codes

Earlier we indicated that a single-error-correcting Hamming code can be made double-error-detecting as well by adding an extra overall parity bit. This results in increasing the block length by one. Such modified codes are known as *extended codes.* Adding an overall parity bit to a code of length $n = 2^m - 1$ results in a codeword whose length is a power of two. This may be advantageous in byte-oriented data handling, or in matching a prespecified field length in a data packet. Extended Reed–Solomon codes are another practical example. As already seen, the natural block length of Reed–Solomon codes is $2^m - 1$ m-bit symbols, and frequently there are reasons to have a block length which is a power of two. Adding an overall parity symbol accomplishes this task. Extended codes may have smaller minimum distance than their original counterparts, but in many instances the minimum distance remains unchanged. An example is the (7, 4) Hamming code and its (8, 4) extended version, both of which have minimum distance 4.

Shortened codes result from the reverse process where we seek to reduce the block length of a basic code. For example, the Reed–Solomon code with 8-bit symbols has a natural block length of 255 symbols. If the encoded data is to be transported in fixed length packets of 240 symbols each, we can set 15 information symbols to zeroes and then delete them before transmission. Shortening can increase minimum distance in general. The shortened code has fewer information bits and the same number of parity bits, so that the error correction capability normalized with respect to block length increases upon shortening.

Product Codes

Elias (26) showed how to combine two block codes into a *product code.* Cyclic product codes were studied by Burton and Weldon (27). Suppose we have an (n_1, k_1) code and an (n_2, k_2) code. Arrange $k = k_1k_2$ data bits in an array of k_1 rows and k_2 columns. Extend each row of k_2 bits into an n_2 bit codeword using the (n_2, k_2) code. Next, extend each of the resulting n_2 columns to n_1 bits using (n_1, k_1) code. The resulting array of $n = n_1n_2$ bits is the product encoding for the original k bits. The rate of the product code is the product of the rates of the constituent codes. If the constituent codes respectively have minimum distances d_1 and d_2, the product code has a minimum distance $d_{\min} = d_1d_2$. Product codes are frequently capable of correcting not only all error patterns of weight $\lfloor (d_1d_2 - 1)/2 \rfloor$ but also many higher weight patterns. However, the simplistic approach of row-wise decoding first, followed by column-wise decoding, may not achieve the full error correction capability of product codes.

Interleaved Coding

The binary symmetric channel models the random error patterns which are bit-to-bit independent. That the bit error probability ϵ is less than 0.5 is the basis for the minimum weight choice of coset leaders (correctable error patterns) in the standard array. In a burst noise channel, errored bit positions tend to cluster together to form error bursts. Codes designed to correct minimum weight error patterns are not directly useful in presence of burst errors. *Interleaved coding* is a technique that allows random-error-correcting codes to effectively combat burst errors. Interleaving renders the burst noise patterns of the channel as apparent random errors to the decoder.

Figure 4 depicts an elementary block interleaver to illustrate the idea. Suppose a burst noise channel is known to generate error bursts spanning at most six consecutive bits. Further, suppose that a (7, 4) Hamming single-error-correcting code is to be used. First we read 24 consecutive data bits row-wise into a 6×4 array, as in Fig. 4. The column size is chosen to be six so as to at least equal the maximum burst length. The row size is chosen to be exactly equal to the number of information digits in the code. Next we extend each row by three more positions by appending the three parity bits appropriate for a Hamming (7, 4) codeword. The extended transmission array is therefore 6×7. The transmission sequence is column-wise—that is, in the sequence 1, 5, 9, 13, 17, 21, 2, 6, 10, 14, . . . (see Fig. 4). The array transmits 42 bits for each 24-bit input. A six-bit error burst may affect the successively transmitted bits 10, 14, 18, 22, 3 and 7, as shown in Fig. 4. Notice that each bit in this error burst belongs to a different Hamming codeword. Thus all such errors are corrected if the error burst does not exceed six bits and there are no other burst error within this span of 42 bits. Suitably sized block interleavers are often effective in burst noise environments.

Concatenated Codes

Consider the following example. Let $n = 8$ and $N = 2^n - 1 = 255$. An (N, K) Reed–Solomon code has $N = 255$ code symbols which can be represented as 8-bit binary words. Transmitted through a binary symmetric channel with bit error probability ϵ, each 8-bit symbol can be in error with probability $\Delta = 1 - (1 - \epsilon)^8$. The code can recover from up to $t_0 = \lfloor (N - K)/2 \rfloor$ symbol errors. The probability of successful decoding is, therefore,

$$P_{s0} = \sum_{i=0}^{t_0} \binom{N}{i} \Delta^i (1-\Delta)^{255-i}$$

We can now *concatenate* the Reed–Solomon code with a Hamming (8, 4) single-error correcting/double-error detecting

1	2	3	4	P1	P2	P3
5	6	7	8	P4	P5	P6
9	10	11	12	P7	P8	P9
13	14	15	16	...	...	...
17	18	19	20	...	...	...
21	22	23	24	...	...	...

Figure 4. An illustration of interleaved coding.

code, as follows. Each 8-bit symbol of the Reed–Solomon code is chosen as a Hamming (8, 4) codeword. To do this, we organize raw data into consecutive blocks of four bits and encode each such into a Hamming (8, 4) codeword. Then each set of K consecutive 8-bit Hamming codewords is encoded into a 255-symbol Reed–Solomon codeword. The probability of a symbol error now becomes smaller, $\delta = 1 - (1 - \epsilon)^8 - 8 \in (1 - \epsilon)^7$, assuming triple and higher errors are negligibly infrequent. Besides, the double-error detection feature identifies the errored symbols in the Reed–Solomon code. With this side information, the Reed–Solomon code can now recover from a greater number of symbol errors, $t_1 = 255 - K$. The probability of successful decoding is now found as

$$P_{s1} = \sum_{i=1}^{t_1} \binom{N}{i} \delta^i (1-\delta)^{255-i}$$

The power of this concatenated coding approach is evident from comparing the above two expressions for the probability of successful decoding. The Reed–Solomon code is the outer code and the Hamming code is the inner code. The inner code cleans up the milder error events and reserves the outer code for the more severe error events. In particular, in a burst noise environment, a long codeword may have some parts completely obliterated by a noise burst while other parts may be affected by occasional random errors. The inner code typically corrects most of the random errors, and the outer Reed–Solomon code combats the burst noise. In most applications the outer code is a suitably sized Reed–Solomon code. The inner code is often a convolutional code (discussed below), though block codes can be used as well, as shown above.

The invention of concatenated coding by Forney (28) was a major landmark in coding theory. Later, Justesen (29) used the concatenation concept to obtain the first constructive codes with rates that do not vanish asymptotically for large block lengths.

Performance Limits of Block Codes

The key performance parameters of a block code are the code rate and the minimum distance. In this section we highlight some of the known bounds on these parameters. The *Hamming bound,* also known as the *sphere packing bound,* is a direct consequence of the following geometrical view of the code space. Let an (n, k) code have minimum distance d. There are 2^k codewords in this code. Around each codeword we can visualize a "sphere" comprising all n-vectors that are within Hamming distance $\lfloor (d-1)/2 \rfloor$ from that codeword. Each such sphere consists of all the n-tuples that result from perturbations of the codeword at the center of the sphere by Hamming weight at most $\lfloor (d-1)/2 \rfloor$. Any two such spheres around two distinct codewords must be mutually exclusive if unambiguous minimum-distance decoding is to be feasible. Thus the total "volume" of all 2^k such mutually exclusive spheres must not exceed the total number of possible n-tuples, 2^n. Thus,

$$2^n \geq 2^k \sum_{i=0}^{t} \binom{n}{i}$$

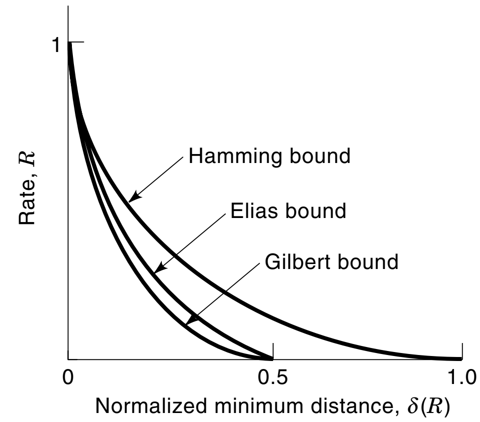

Figure 5. Bounds on the minimum distance of block codes.

This can be expressed in the following form of the Hamming upper bound on code rate R:

$$R \leq 1 - \frac{1}{n} \log \sum_{i=0}^{t} \binom{n}{i}$$

Asymptotically for large n, this reduces to the form

$$R \leq 1 - H_2\left(\frac{\delta(R)}{2}\right)$$

where $H_2(x) = -x \log x - (1 - x) \log(1 - x)$ is the binary entropy function; $\delta(R)$ is the largest value of the normalized minimum distance of a rate-R code as the block length n goes to vary large values (i.e., $\lim \sup_{N\to\infty} d_{\min}/n$); and the logarithm is to the base 2 so that R is in bits of information per binary digit.

The Hamming upper bound asserts that for a given $\delta(R)$, no code can exceed the rate given by the bound above. The *Gilbert bound,* on the other hand, is a constructive bound which states that it is possible, for a given $\delta(R)$, to construct codes with rates at least as large as the value R specified by the bound. The asymptotic Gilbert bound states that

$$R \geq 1 - H_2(\delta(R))$$

The *Elias bound* is a tighter upper bound on the feasible code rates compared to the Hamming bound. In its asymptotic form the Elias bound is stated as follows:

$$\delta(R) \leq 2\lambda_R(1 - \lambda_R)$$

where

$$R = 1 - H_2(\lambda_R)$$

These bounds are shown in Fig. 5. The feasibility region of "good" block codes lies between the Gilbert and Elias bounds. Hamming bound originally appeared in Ref. 10, and the Gilbert bound in Ref. 30. The Elias bound was first developed by Elias *circa* 1959 but appeared in print only in 1967 paper by Shannon, Gallager, and Berlekamp (see Ref. 5, p. 3). Proofs for these bounds are found in many coding theory books (e.g., Ref. 3). It had been conjectured for some time that the Gilbert bound was asymptotically tight—that is, that it was an upper bound as well as a lower bound and that all long, good codes would asymptotically meet the Gilbert bound exactly. This perception was disproved for nonbinary codes by the work of Tsfasman et al. (31). Also McEliece et al. (32) obtained some improvements on the Elias bound. See also Ref. 33 for tabulations of the best-known minimum distances of block codes.

CONVOLUTIONAL CODES

Convolutional Encoders

Convolutional codes were originally proposed by Elias (34). Probabilistic search algorithms were developed by Fano (35) and Wozencraft and Reiffan (36) as practical decoding algorithms. Massey (37) proposed the use of threshold decoding for convolutional codes as a simpler though less efficient alternative. The Viterbi algorithm (38) was developed later as an efficient decoder for short convolutional codes. We will briefly outline Wozencraft's sequential decoding and the Viterbi algorithm, after examining the basic structure of convolutional codes.

Convolutional coding is based on the notion of passing an arbitrarily long sequence of input data bits through a linear sequential machine whose output sequence has memory properties and consequent redundancies that allow error correction. A linear sequential machine produces each output symbol as a linear function of the current input and a given number of the immediate past inputs, so that the output symbols have "memory" or temporal correlation. Certain symbol patterns are more likely than others, and this allows error correction based on maximum likelihood principles. The output of the linear sequential machine is the convolution of its impulse reponse with the input bit stream, hence the name. Block codes and convolutional codes are traditionally viewed as the two major classes error correction codes, although we will recognize shortly that it is possible to characterize finite length convolutional codes in a formalism similar to that used to describe block codes.

A simple convolutional encoder is shown in Fig. 6. For every input bit, the encoder produces two output bits. The code rate is hence $\frac{1}{2}$. (More generally, a convolutional encoder may accept k input bits at a time and produce n output bits, implementing a rate k/n code.) The output of the encoder in Fig. 6 is a function of the current input and the two previous inputs. One input bit is seen to affect three successive pairs of output bits. We say that the *constraint length* of the code is therefore $K = 6$. There are other definitions of the constraint length, as the number of consecutive input bits that affect a given

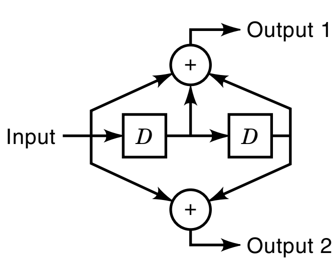

Figure 6. A convolutional encoder.

output ($K = 3$) or the minimum number of delay elements needed to implement the encoder ($K = 2$).

It must be noted that Fig. 6 could have been redrawn with only two memory elements to store the two previous bits; the current input bit could be residing on the input line. The memory order of the encoder in Fig. 6 is thus only two, and the encoder output is determined by the state of the encoder (which is the content of the two memory registers) and by the new input bit. Whether we use an extra memory register to hold the incoming new bit or not is similar in spirit to the distinction between the Moore and the Mealy machines in the theory of finite-state sequential machines (39).

The impulse response of the encoder at the upper output of the convolutional encoder in Fig. 6 is '1 1 1' and that at the lower output line is '1 0 1'. The output sequences at these lines are therefore the discrete convolutions of the input stream with these impulse responses. The following infinite-dimensional generator matrix represents the mapping of the infinite input sequence ($x_0, x_1, x_2, \ldots$) into the infinite output sequence ($y_0, y_1, y_2, \ldots$) where y_{2n} and y_{2n+1} are the two output bits corresponding to input bit x_n:

$$
x_0 x_1 x_2 \ldots] \left[\begin{matrix}
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots
\end{matrix}\right.
$$

$$
\left.\begin{matrix}
0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
1 & 1 & 0 & 0 & 0 & 0 & \cdots \\
1 & 0 & 1 & 1 & 0 & 0 & \cdots \\
1 & 1 & 1 & 0 & 1 & 1 & \cdots \\
0 & 0 & 1 & 1 & 1 & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & 1 & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots
\end{matrix}\right] = [y_0 y_1 y_2 \ldots
$$

Also, in terms of the impulse response polynomials $G_1(D) = 1 + D + D^2$ and $G_2(D) = 1 + D^2$, respectively, for the upper and lower output lines in Fig. 6, we can relate the input polynomial $X(D)$ to the respective output polynomials as

$$Y_i(D) = X(D)G_i(D), \qquad i = 1, 2$$

However, these matrix and polynomial algebraic approaches are not as productive here as they were for the block codes. More intuitive insight into the nature of convolutional codes can be furnished in terms of its tree and trellis diagrams.

Trees, State Diagrams, and Trellises

The most illuminating representation of a convolutional code is in terms of the associated tree diagram. The encoding process starts at the root node of a binary tree, as shown in Fig.

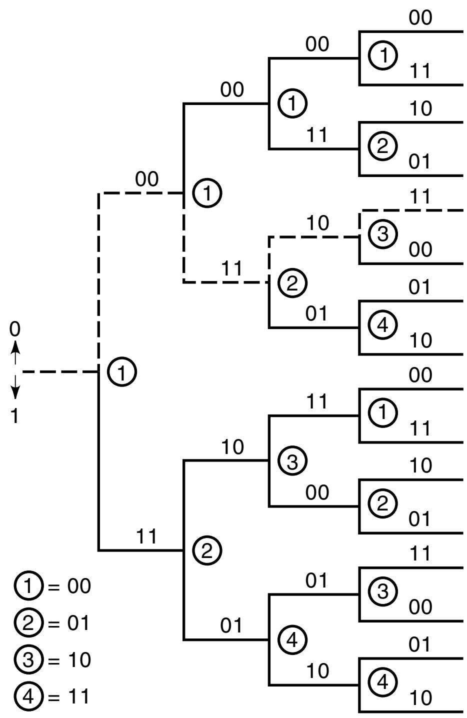

Figure 7. A tree diagram for the convolutional encoder in Fig. 6.

7 for the encoder in Fig. 6. Each node spawns two branches. Each successive input bit causes the process to move to one of the next higher level nodes. If the input bit is a zero, the upper branch is taken, otherwise the lower one. The labeling on each branch shows the bit pair produced at the output for each branch. Tracing an input sequence through the tree, the concatenation of the branch labels for that path produces the corresponding codeword.

Careful inspection of the tree diagram in Fig. 7 reveals a certain repetitive structure depending on the "state" of the encoder at each tree node. The branching patterns from any two nodes with identical states are seen to be identical. This allows us to represent the encoder behavior most succinctly in terms of a state transition diagram in Fig. 8. The state of the encoder is defined as the contents of the memory elements at any time. The encoder in Fig. 7 has four states, 1 = 00, 2 = 01, 3 = 10 and 4 = 11. The solid lines in Fig. 8 indicate state transitions caused by a zero input, and the dotted lines indicate input one. The labels on the branches are the output bit pairs, as in the tree diagram in Fig. 7.

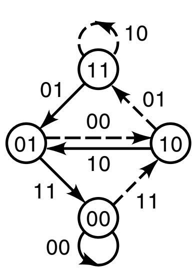

Figure 8. The state transition diagram for the convolutional encoder in Fig. 6.

Data sequences drive the encoder through various sequences of state transitions. The pattern of all such possible state transition trajectories in time is known as a *trellis diagram.* In Fig. 9 we have the trellis diagram for an encoder that starts in state 00 and encodes a 7-bit input sequence whose last two bits are constrained to be zeroes. This constraint, useful in Viterbi decoding to be described below, terminates all paths in state 00. The trellis diagram in Fig. 9 thus contains 2^5 distinct paths of length 7 beginning and ending in state 00.

Weight Distribution for Convolutional Codes

An elegant method for finding the weight distribution of convolutional codes is to redraw the state transition diagram such as in Fig. 8, in the form shown in Fig. 10 with the all-zero state (00 in our example) split into two, a starting node and an ending node. To each directed path between two states, we assign a "gain" W^i, where W is a dummy variable and the exponent i is the Hamming weight of the binary sequence emitted by the encoder upon making the indicated state transition. For example, in Fig. 10, the transition from 1 to 2 causes the bit pair 11 to be emitted, with Hamming weight $i = 2$, so that the gain is W^2. In transitions that emit 01 or 10, the gain is W and in the case where 00 is emitted, the gain is $W^0 = 1$. We can now use a "signal flow graph" technique due to Mason (40) to obtain a certain "transfer function" of the encoder. In the signal flow graph method, we postulate an input signal S_{in} at the starting state and compute the output signal S_{out} at the ending state, using the following relations among signal flow intensities at the various nodes:

$$\begin{aligned} S_{\text{out}} &= S_2 W^2 \\ S_2 &= (S_3 + S_4)W \\ S_4 &= (S_3 + S_4)W \\ S_3 &= S_{\text{in}} W^2 \end{aligned}$$

The transfer function $T(W) = S_{\text{out}}/S_{\text{in}}$ can be readily found to be

$$T(W) = \frac{W^5}{(1-2W)} = \sum_{i=0}^{\infty} 2^i W^{5+i} = W^5 + 2W^6 + 4W^7 + \cdots$$

Each term in the above series corresponds to a set of paths of a given weight. The coefficient 2^i gives the number of paths of weight $5 + i$. There is exactly one path of weight 5, two paths of weight 6, four of weight 7, and so on. There are no paths of weight less than 5. The path with weight 5 is seen to be the closest in Hamming distance to the all-zero codeword. This distance is called the *free distance,* d_{free}, of the code. In the present example, $d_{\text{free}} = 5$. The free distance of a convolutional code is a key parameter in defining its error correction, as will be seen in the next section.

Maximum Likelihood (Viterbi) Decoding for Convolutional Codes

Each path in a trellis diagram corresponds to a valid code sequence in a convolutional code. A received sequence with bit errors in it will not necessarily correspond exactly to any one particular trellis path. The Viterbi algorithm (38) is a

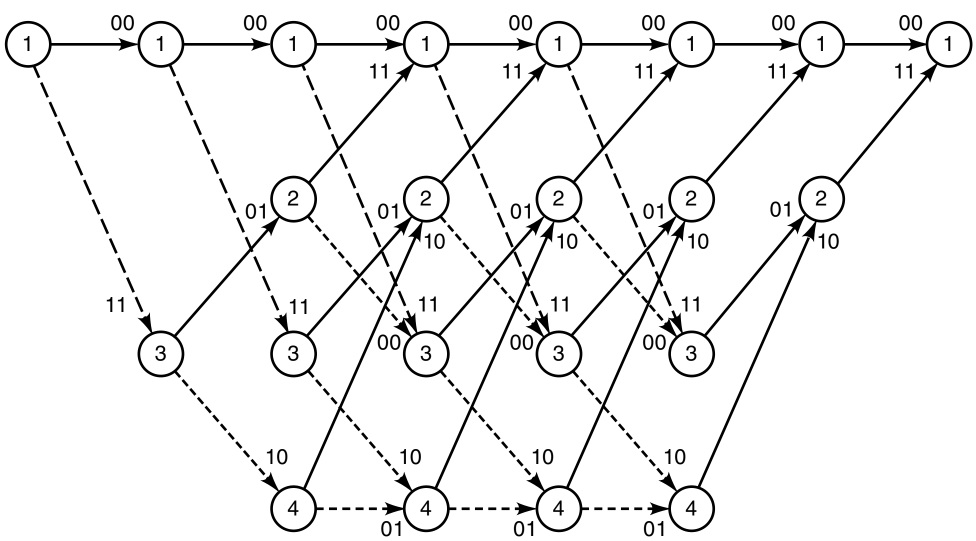

Figure 9. The trellis diagram for the convolutional encoder in Fig. 6.

computationally efficient way for discovering the most likely transmitted sequence for any given received sequence of bits. With reference to the trellis diagram in Fig. 9, suppose that we have received the sequence 11 01 10 01 01 10 11. Starting in state 1 at time $t = 0$, the trellis branches out to states 1 or 2 in time $t = 1$, and from there to all four states 1, 2, 3, 4, in time $t = 2$. At this point there is exactly one unique path to each of the four current possible states staring from state 1. In order to reach state 1 at time $t = 2$, the trellis path indicates the transmitted prefix sequence 00 00 which is at a Hamming distance three from the actual received prefix 11 01. The path reaching state 2 in time $t = 2$ in the trellis diagram similarly corresponds to the transmitted prefix sequence 11 01 which is seen to be at Hamming distance zero from the corresponding prefix of the received sequence. Similarly we can associate Hamming distances 3 and 2 respectively to the paths reaching states 3 and 4 in time $t = 2$ in the trellis diagram.

Now we extend the trellis paths to time $t = 3$. Each state can be reached at time $t = 3$ along two distinct paths. For instance, in order to reach state 1 in time $t = 3$, the encoder could have made a 1 to 1 transition, adding an incremental Hamming distance of one to the previous cumulative value of three; or it could have made the 2 to 1 transition, adding one unit of Hamming weight to the previous value of zero. Thus at time $t = 3$, there are two distinct paths merging at state 1: the state sequence 1–1–1–1 with a cumulative Hamming distance of four from the given received sequence, and the sequence 1–3–2–1 with a cumulative Hamming distance of one. Since the Hamming weights of the paths are incremental, for any path emanating from state 1 at time $t = 3$, the prefix with the lower cumulative distance is clearly the better choice. Thus at this point we discard the path 1–1–1–1 from further consideration and retain the unique *survivor path* 1–3–2–1 in association with state 1 at the current time. Similarly we explore the two contending paths converging at the other three states at this time ($t = 3$) and identify the minimum distance (or maximum likelihood, for the BSC) survivors for each of those states.

The procedure now iterates. At each successive stage, we identify the survivor paths for each state. If the code sequence were infinite, we would have four infinitely long parallel path traces through the trellis in our example. In order to choose one of the four as the final decoded sequence, we require the encoder to "flush out" the data with a sequence of zeroes, two in our example. The last two zeroes in the seven-bit input data to the encoder cause the trellis paths to converge to state 1 or 2 at time $t = 6$ and to state 1 at $t = 7$. By choosing the survivors at these states, we finally have a complete trellis path starting from state 1 at time $t = 0$ and ending in state 1 at time $t = 7$. The output labels of the successive branches along this path gives the decoder's maximum likelihood estimate of the transmitted bits corresponding to the received sequence.

The average bit error rate of the Viterbi decoder, P_b, can be shown to be bounded by an exponential function of the free distance of the code, as below:

$$P_b \approx N_{d_{\text{free}}}[2\sqrt{\epsilon(1-\epsilon)}]^{d_{\text{free}}} \approx N_{d_{\text{free}}}[2\sqrt{\epsilon}]^{d_{\text{free}}}$$

This applies to codes that accept one input bit at a time, as in Fig. 6. $N_{d_{\text{free}}}$ is the total number of nonzero information bits on all trellis paths of weight d_{free}, and it can in general be found via an extension of the signal flow transfer function method outlined above. The parameter ϵ is the BSC error probability and is assumed to be very small in the above approximation.

The Viterbi algorithm needs to keep track of only one survivor path per state. The number of states, however, is an exponential function of the memory order. For short convolutional codes of modestly sized state space, the Viterbi algo-

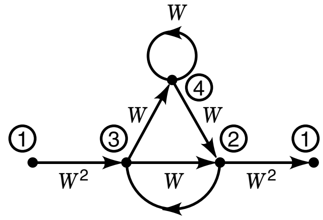

Figure 10. The signal flow graph for the convolutional encoder in Fig. 6.

rithm is an excellent choice for decoder implementation. A memory order of 7 or 8 is typically the maximum feasible. This, in turn, limits the free distance and hence the bit error probability. For long convolutional codes, the survivor path information storage required per state becomes large. In practice, we may choose to retain only some most recent segment of the history of each survivor path. The resulting "truncated" Viterbi algorithm is no longer the theoretically ideal maximum likelihood decoder, though its performance is usually close to the ideal decoder. All these considerations restrict the application of the Viterbi algorithm to short convolutional codes with small constraint lengths. Within these limitations, however, the Viterbi algorithm affords excellent performance.

Sequential Decoding for Convolutional Codes

Tree diagrams lead to one viable strategy for decoding convolutional codes. Given a received sequence of bits (possibly containing errors), the decoder attempts to map it to one path along the tree, proceeding node by node and keeping track of the cumulative Hamming distance of the path from the received sequence. Along a wrong path, the cumulative Hamming distance exceeds a preset threshold after a few nodes, whereupon the decoder backtracks to the previous node and explores another path. The time to decode any given sequence in this scheme is a random variable, but its expected value remains bounded for code rates below a number $R_{\text{comp}} < C$, where R_{comp} is the *computational cutoff rate* and C is the channel capacity. This technique, known as sequential decoding, is an appropriate technique for decoding very long convolutional codes.

A sequential decoder executes a random number of computations to decode a received sequence—unlike the Viterbi decoder, which executes a fixed number of computations per code sequence. This can be a strength or a weakness, depending on the average noise intensity. If the noise level is high, the sequential decoder typically has to explore many false paths before it discovers the correct path. But the Viterbi algorithm produces an output after a fixed number of computations, possibly faster than the sequential decoder. On the other hand, if the noise level is low, the Viterbi algorithm still needs to execute all of its fixed set of computations whereas the sequential decoder will typically land on the right tree path after only a few trials. Also, sequential decoding is preferred in applications where long codes are needed to drive the postdecoding error probability to extremely low values. In such cases, complexity considerations eliminate Viterbi algorithm as a viable choice.

An efficient approach to implementing sequential decoding is the *stack algorithm*. The key idea here is that the previously explored paths and their likelihood metrics can be stored in a stack ordered according to the likelihood metric value, with the most likely path at the top. The topmost path is then extended to the set of branches extending from that node, metrics are recomputed, and the stack is updated with the new information.

ADDITIONAL TOPICS

Burst Noise Channels

In the foregoing discussion, we had mostly assumed the binary symmetric channel model which was the basis for minimum distance decoding. Burst error channels are another important class of transmission channels encountered in practice, both in wireline and wireless links. Errored bit positions tend to cluster together in such channels, making direct application of much of the foregoing error correction codes futile in such cases. We have already mentioned interleaved coding as a practical method for breaking the error clusters into random patterns and then using random-error correcting codes. Also we noted that Reed–Solomon codes have an intrinsic burst error correction capability. In addition, there have been error correction codes specifically developed for burst noise channels. For a detailed treatment of this subject, see, for example, Ref. 8, Chap. 9.

Intersymbol Interference Channels and Precoding

Binary data are transmitted by mapping the 0's and 1's into baseband or radio-frequency (RF) pulses. In a bandwidth-limited channel, the channel response waveform corresponding to one input pulse tends to overlap those of succeeding pulses, if the input pulse rate is high. This *intersymbol interference* (ISI) can be controlled by appropriately shaping the input spectrum by *precoding* the input pulse waveform. By suitably constraining the 0/1 transition patterns, it becomes possible to receive the input bit stream despite the overlap of the pulse response waveforms. This technique has been important in high-speed modem designs for the wireline channel. Because of the recent interest in digital subscriber lines, there has been much activity in this area. We cite Ref. 41 as an example of recent work and for pointers to earlier work in this important area.

Synchronization Codes

Coding techniques described so far implicitly assume synchronization; that is, the decoder knows the exact times when one codeword ends and the next begins, in a stream of binary data. In real life this of course cannot be assumed. Codes that can self-synchronize are therefore important. Key results in this direction is summarized in standard coding theory sources such as Ref. 4. However, the practical use of these synchronization codes does appear to be limited, compared to more advanced timing and synchronization techniques used in modern digital networks.

Soft Decision Decoding

The actual inputs and outputs of the physical channel are analog waveforms. The demodulator processes the noisy waveform output of the physical channel and furnishes a noisy estimate of the currently transmitted bit or symbol. A *hard decision* is made at the demodulator output when a threshold device maps the noisy analog data into a 0 or a 1 (in the binary case). Instead, we can retain the analog value (or a finely quantized version of it) and then make an overall decision about the identity of an entire codeword from these *soft decision* data. Clearly, the soft decision data retain more information, and hence the overall decision made on an entire codeword can be expected to be more reliable than the concatenation of bit-by-bit hard decisions. Analysis and practical implementations have borne out this expectation, and soft decision decoding enables achievement of the same bit error rate with a lower signaling power requirement than that for hard decision decoding. Many recent text books on digital communications (e.g., Ref. 42) contain details of this approach.

Combined Coding and Modulation

As mentioned earlier, coding and modulation have traditionally developed in mutual isolation. Ungerboeck (43) proposed the idea that redundancy for error correction coding may be embedded into the design of modulation signal constellations, and combined decoding decisions may be based on the Euclidean distance between encoded signal points rather than on Hamming distance. The approach has been found to be capable of significant improvements in the performance of coded communications. For more details on this topic, see Ref. 44 or one of the more recent textbooks in digital communications such as Ref. 42.

Turbo Codes

The discovery of "turbo codes" (45) is perhaps the most spectacular event in coding research in recent times. Turbo codes have made it possible to approach the ultimate Shannon limits for communication much more closely than was previously possible. Turbo codes are essentially a battery of parallel concatenated encoders. The outputs of these component encoders are merged by interleaving, and they are punctured as needed to get the desired code rate. Relatively simple, iterative soft decision decoding methods provide surprisingly superior performance.

APPLICATIONS

Data transmission coding is intrinsically an applications-oriented discipline, its deep mathematical foundations notwithstanding. Early work by Hamming (10) and others were quickly applied to computer data storage and transmission. The minimum-distance decoding approach is ideally suited ro random, independent-error environments such as found in space communications, where coding applications registered some early success (e.g., see Chapter 3 of Ref. 23). The markedly clustered or bursty nature of error patterns in terrestrial wireline and radio channels initially limited coding applications in this arena to only error detection and retransmission (8, Chapter 15), and not forward error correction. Cyclic redundancy check (CRC) codes are cyclic codes used for error detection only, and they are ubiquitous in modern data transmission protocols. Also, more recently, the needs of the high-speed modems created more opportunities for error correction applications. Many specific codes have lately been adopted into international standards for data transmission. Chapters 16 and 17 in Ref. 8 furnish an excellent summary of applications of block and convolutional codes, respectively. Applications of Reed–Solomon codes to various situations are well documented in Ref. 23, including its use in compact disc players, deep space communications, and frequency hop spread spectrum packet communications.

The advent of the broadband integrated service digital network (B-ISDN) has created ample instances of coding applications (46). The asynchronous transfer mode (ATM) adaptation layer 1 (AAL-1) has a standardized option for the use of Reed–Solomon coding for recovery from ATM cell losses (47, p. 75). The recently adopted high-definition television (HDTV) standards use Reed–Solomon coding for delivery of compressed, high-rate digital video (48). Almost all of the recent digital wireless technologies, such as GSM, IS-54 TDMA, IS-95 CDMA, cellular digital packet data (CDPD), and others (49), have found it advantageous to make use of error correction coding to mitigate the excessive noisiness of the wireless channel.

In summary, over the past 50 years following the inception of information theory (1), not only has the art of data transmission codes matured into a variety of applications technologies, but also we are remarkably close to the ultimate theoretical limits of coding performance predicted in Ref. 1.

BIBLIOGRAPHY

1. C. E. Shannon, A mathematical theory of communications, *Bell Syst. Tech. J.,* **27**: 379–423, 623–656, 1948.
2. R. G. Gallager, *Information Theory and Reliable Communication,* New York: Wiley, 1968.
3. E. R. Berlekamp, *Algebraic Coding Theory,* New York: McGraw-Hill, 1989.
4. W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes,* 2nd ed., Cambridge, MA: MIT Press, 1972.
5. E. R. Berlekamp, *Key Papers in the Development of Coding Theory,* New York: IEEE Press, 1974.
6. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes,* Amsterdam, The Netherlands: North-Holland, 1977.
7. R. E. Blahut, *Theory and Practice of Error Control Codes,* Reading, MA: Addison-Wesley, 1983.
8. S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications,* Englewood Cliffs, NJ: Prentice-Hall, 1983.
9. A. M. Michelson and A. H. Levesque, *Error Control Techniques for Digital Communications,* New York: Wiley, 1985.
10. R. W. Hamming, Error detecting and error correcting codes, *Bell Syst. Tech. J.,* **29**: 147–160, 1950.
11. D. E. Slepian, A class of binary signaling alphabets, *Bell Syst. Tech. J.,* **35**: 203–234, 1956.
12. F. J. MacWilliams, A theorem on the distribution of weights in a systematic code, *Bell Syst. Tech. J.,* **42**: 79–94, 1963.
13. E. Prange, Cyclic error-correcting codes in two symbols, *AFCRC-TN-57-103,* Air Force Cambridge Research Center, Cambridge, MA, 1957.
14. R. C. Bose and D. K. Ray-Chaudhuri, On a class of error correcting binary group codes, *Inf. Control,* **3**: 68–79, 1960.
15. A. Hocquenghem, Codes correcteurs d'erreurs, *Chiffres,* **2**: 147–156, 1959, in French.
16. W. W. Peterson, Encoding and decoding procedures for the Bose–Chaudhuri codes, *IRE Trans. Inf. Theory,* **6**: 459–470, 1960.
17. D. C. Gorenstein and N. Zierler, A class of error-correcting codes in p^m symbols, *J. Soc. Ind. Appl. Math (SIAM),* **9**: 207–214, 1961.
18. R. T. Chien, Cyclic decoding procedure for the BCH codes, *IEEE Trans. Inf. Theory,* **10**: 357–363, 1964.
19. E. R. Berlekamp, On decoding binary BCH codes, *IEEE Trans. Inf. Theory,* **11**: 577–580, 1965.
20. J. L. Massey, Shift register synthesis and BCH decoding, *IEEE Trans. Inf. Theory,* **15**: 122–127, 1969.
21. R. E. Blahut, Transform techniques for error control codes, *IBM J. Res. Develop.,* **23**: 299–315, 1979.
22. I. S. Reed and G. Solomon, Polynomial codes over certain finite fields, *J. Soc. Ind. Appl. Math. (SIAM),* **8**: 300–304, 1960.

23. S. B. Wicker and V. K. Bhargava, *Reed–Solomon Codes and Their Applications,* Piscataway, NJ: IEEE Press, 1994.
24. A. Tietvainen, A short proof for the nonexistence of unknown perfect codes over GF(q), $q > 2$, *Ann. Acad. Sci. Fenn. A,* **580**: 1–6, 1974.
25. M. J. E. Golay, Binary coding, *IRE Trans. Inf. Theory,* **4**: 23–28, 1954.
26. P. Elias, Error-free coding, *IRE Trans. Inf. Theory,* **4**: 29–37, 1954.
27. H. O. Burton and E. J. Weldon, Jr., Cyclic product codes, *IRE Trans. Inf. Theory,* **11**: 433–440, 1965.
28. G. D. Forney, Jr., *Concatenated Codes,* Cambridge, MA: MIT Press, 1966.
29. J. Justesen, A class of constructive asymptotically algebraic codes, *IEEE Trans. Inf. Theory,* **18**: 652–656, 1972.
30. E. N. Gilbert, A comparison of signaling alphabets, *Bell Syst. Tech. J.,* **31**: 504–522, 1952.
31. M. A. Tsfasman, S. G. Vladut, and T. Zink, Modular curves, Shimura curves and Goppa codes which are better than the Varsharmov–Gilbert bound, *Math. Nachr.,* **109**: 21–28, 1982.
32. R. J. McEliece et al., New upper bounds on the rate of a code via the Delsarte–MacWilliams inequalities, *IEEE Trans. Inf. Theory,* **23**: 157–166, 1977.
33. T. Verhoeff, An updated table of minimum-distance bounds for binary linear codes, *IEEE Trans. Inf. Theory,* **33**: 665–680, 1987.
34. P. Elias, Coding for noisy channels, *IRE Conv. Rec.,* **4**: 37–47, 1955.
35. R. M. Fano, A heuristic discussion of probabilistic decoding, *IEEE Trans. Inf. Theory,* **9**: 64–74, 1963.
36. J. M. Wozencraft and B. Reiffan, *Sequential Decoding,* Cambridge, MA: MIT Press, 1961.
37. J. L. Massey, *Threshold Decoding,* Cambridge, MA: MIT Press, 1963.
38. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inf. Theory,* **13**: 260–269, 1967.
39. G. D. Forney, Jr., Convolutional codes I: Algebraic structure, *IEEE Trans. Inf. Theory,* **16**: 720–738, 1970.
40. S. J. Mason, Feedback theory—Further properties of signal flow graphs, *Proc. IRE,* **44**: 920–926, 1956.
41. R. F. H. Fischer and J. B. Huber, Comparison of precoding schemes for digital subscriber lines, *IEEE Trans. Commun.,* **45**: 334–343, 1997.
42. S. G. Wilson, *Digital Modulation and Coding,* Upper Saddle River, NJ: Prentice-Hall, 1996.
43. G. Ungerboeck, Channel coding with amplitude/phase modulation, *IEEE Trans. Inf. Theory,* **28**: 55–67, 1982.
44. G. Ungerboeck, *IEEE Commun. Mag.,* **25** (2): 12–21, 1987.
45. C. Berrou and A. Glavieux, Near-optimum error correcting coding and decoding: Turbo codes, *IEEE Trans. Commun.,* **44**: 1261–1271, 1996.
46. E. Ayanoglu, R. D. Gitlin, and N. C. Oguz, Performance improvement in broadband networks using a forward error correction for lost packet recovery, *J. High Speed Netw.,* **2**: 287–304, 1993.
47. C. Partridge, *Gigabit Networks,* Reading, MA: Addison-Wesley, 1994.
48. T. S. Rzeszewski, *Digital Video: Concepts and Applications across Industries,* Piscataway, NJ: IEEE Press, 1995.
49. T. S. Rappaport, *Wireless Communications: Principles and Practice,* Upper Saddle River, NJ: Prentice-Hall, 1996.

George Thomas
University of Southwestern
Louisiana