CONVOLUTIONAL CODES

RICHARD D. WESEL
University of California at
Los Angeles
Los Angeles, California

1. INTRODUCTION

Convolutional codes represent one technique within the general class of channel codes. Channel codes (also called *error-correction codes*) permit reliable communication of an information sequence over a channel that adds noise, introduces bit errors, or otherwise distorts the transmitted signal. Elias [1,2] introduced convolutional codes in 1955. These codes have found many applications, including deep-space communications and voiceband modems. Convolutional codes continue to play a role in low-latency applications such as speech transmission and as constituent codes in Turbo codes. Two reference books on convolutional codes are those by Lin and Costello [3] and Johannesson and Zigangirov [4].

Section 2 introduces the shift-register structure of convolutional encoders including a discussion of equivalent encoders and minimal encoders. Section 3 focuses on the decoding of convolutional codes. After a brief mention of the three primary classes of decoders, this section delves deeply into the most popular class, Viterbi decoders. This discussion introduces trellis diagrams, describes the fundamental add-compare-select computation, compares hard and soft decoding, and describes the suboptimal (but commonly employed) finite traceback version of Viterbi decoding.

Section 4 defines the free distance of a convolutional code and describes how free distance may be computed by a specialized application of the Viterbi algorithm. This procedure also yields an analytic lower bound on the decision depth that should be used for finite-traceback decoding. Catastrophic encoders are also discussed in this section. Section 5 describes the generating function that enumerates all the paths associated with error events in the decoder trellis. This section then gives union bounds on bit error rate that are computed from the generating function. Section 6 provides some final remarks regarding the effective blocklength of convolutional codes and their role today.

2. ENCODER STRUCTURE

As any binary code, convolutional codes protect information by adding redundant bits. A rate-k/n convolutional encoder processes the input sequence of k-bit information symbols through one or more binary shift registers (possibly employing feedback). The convolutional encoder computes each n-bit symbol ($n > k$) of the output sequence from linear operations on the current input symbol and the contents of the shift register(s). Thus, a rate k/n convolutional encoder processes a k-bit input symbol and computes an n-bit output symbol with every shift register update. Figures 1 and 2 illustrate feedforward and feedback encoder implementations of a rate-$\frac{1}{2}$ code. Section 2.1 explores the similarities and differences between feedforward and feedback encoders by examining their state diagrams.

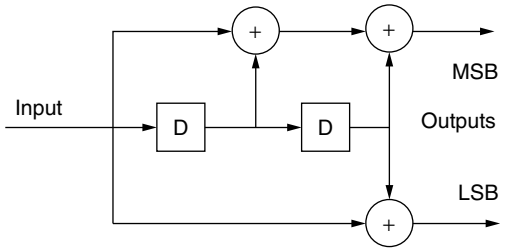

Figure 1. Rate-$\frac{1}{2}$ feedforward convolutional encoder with two memory elements (four states). MSB and LSB refer to the most and least significant bits, respectively.

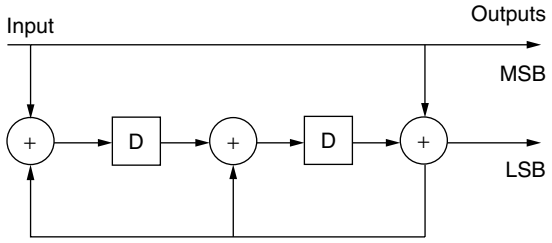

Figure 2. Rate-$\frac{1}{2}$ feedback convolutional encoder with two memory elements (four states).

2.1. Equivalent Encoders

Convolutional encoders are finite-state machines. Hence, state diagrams provide considerable insight into their behavior. Figures 3 and 4 provide the state diagrams for the encoders of Figs. 1 and 2, respectively. The states are labeled so that the least significant bit is the one residing in the leftmost memory element of the shift register. The branches are labeled with the 1-bit (single-bit) input and the 2-bit output separated by a comma. The most significant bit (MSB) of the two-bit output is the bit labeled MSB in Figs. 1 and 2.

If one erases the state labels and the single-bit input labels, the remaining diagrams for Figs. 3 and 4 (labeled with only the 2-bit outputs) would be identical. This illustrates that the two encoders are equivalent in the

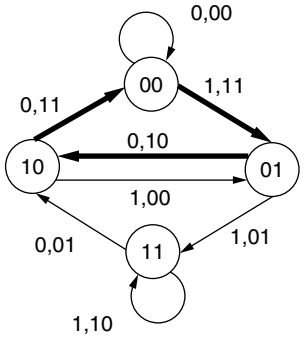

Figure 3. State diagram for rate-$\frac{1}{2}$ feedforward convolutional encoder of Fig. 1.

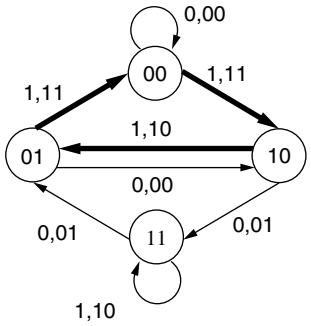

Figure 4. State diagram for rate-$\frac{1}{2}$ feedback convolutional encoder of Fig. 2.

sense that both encoders produce the same set of possible output sequences (or codewords). Strictly speaking, a code refers to the list of possible output sequences without specifying the mapping of inputs sequences to output sequences. Thus, as in the above example, two equivalent encoders have the same set of possible output sequences, but may implement different mappings from input sequences to output sequences. In the standard convolutional coding application of transmission over additive white Gaussian noise (AWGN) with Viterbi decoding, such encoders give similar BER performance, but the different mappings of inputs to outputs do lead to small performance differences.

The three-branch paths emphasized with thicker arrows in Figs. 3 and 4 are each the shortest nontrivial (i.e., excluding the all-zeros self-loop) loop from the all-zeros state back to itself. Notice that for Fig. 3, the state diagram corresponding to the feedforward encoder, this loop requires only a single nonzero input. In contrast, for the state diagram corresponding to Fig. 4, this loop requires three nonzero inputs. In fact, for Fig. 4 no nontrivial loop from the all-zeros state to itself requires fewer than two nonzero inputs. Thus the feedforward shift register has a finite impulse response, and the feedback shift register has an infinite impulse response.

This difference is not particularly important for convolutional codes decoded with Viterbi, but it is extremely important to convolutional encoders used as constituents in Turbo codes, which are constructed by concatenating convolutional codes separated by interleavers. Only feedback encoders (with infinite impulse responses) are effective constituents in Turbo codes. Thus, equivalent encoders can produce dramatically different performance as constituents in Turbo codes, depending on whether or not they meet the requirement for an infinite impulse response.

2.2. Minimal Encoders

A practical question to ask about a convolutional encoder is whether there is an equivalent encoder with fewer memory elements. This question may be answered by performing certain diagnostic computations on the encoder matrix. Furthermore, if the encoder is not minimal, it may be easily "repaired" yielding an encoder that is equivalent but requires fewer memory elements. Forney's classic paper [5] treats this fundamental area of convolutional coding theory elegantly. More recently, Johannesson and Wan [6] extended Forney's results by taking a linear algebra approach. This fascinating area of convolutional code theory is important to convolutional code designers, but less so for "users." Any code published in a table of good convolutional codes will be minimal.

3. DECODING CONVOLUTIONAL CODES

Convolutional code decoding algorithms infer the values of the input information sequence from the stream of received distorted output symbols. There are three major families of decoding algorithms for convolutional codes: sequential, Viterbi, and maximum a posteriori (MAP). Wozencraft proposed sequential decoding in 1957 [7]. Fano in 1963 [8] and Zigangirov in 1966 [9] further developed sequential decoding. See the book by Johannesson and Zigangirov [4] for a detailed treatment of sequential decoding algorithms. Viterbi originally described the decoding algorithm that bears his name in 1967 [10]. See also Forney's work [11,12] introducing the trellis structure and showing that Viterbi decoding is maximum-likelihood in the sense that it selects the sequence that makes the received sequence most likely.

In 1974, Bahl et al. [13] proposed MAP decoding, which explicitly minimizes bit (rather than sequence) error rate. Compared with Viterbi, MAP provides a negligibly smaller bit error rate (and a negligibly larger sequence error rate). These small performance differences require roughly twice the complexity of Viterbi, making MAP unattractive for practical decoding of convolutional codes. However, MAP decoding is crucial to the decoding of Turbo codes. For the application of MAP decoding to Turbo codes, see the original paper on Turbo codes by Berrou et al. [14] and Benedetto et al.'s specific discussion of the basic turbo decoding module [15].

When convolutional codes are used in the traditional way (not as constituents in Turbo codes), they are almost always decoded using some form of the Viterbi algorithm, and the rest of this section focuses on describing Viterbi. The goal of the Viterbi algorithm is to find the transmitted sequence (or codeword) that is closest to the received sequence. As long as the distortion is not too severe, this will be the correct sequence.

3.1. Trellis Diagrams

The state diagrams of Figs. 3 and 4 illustrate what transitions are possible from a particular state regardless of time. In contrast, trellis diagrams use a different branch for each different symbol time. As a result, trellis diagrams more clearly illustrate long trajectories through the states. Figure 5 shows one stage (one symbol time) of the trellis diagram associated with the rate-$\frac{1}{2}$ feedforward encoder of Figs. 1 and 3. Each column of states in the trellis diagram includes everything in the original state diagram. All the branches emanating from states in a particular column are incident on the states in the adjacent column to the

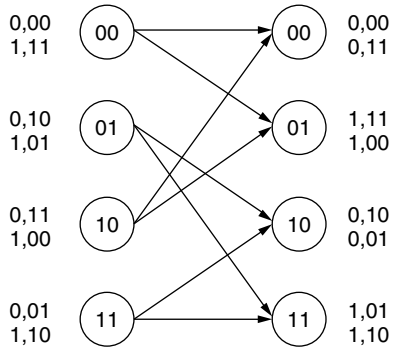

Figure 5. One stage of the trellis diagram for rate-$\frac{1}{2}$ feedforward convolutional encoder of Figs. 1 and 3.

right. In other words, each state transition in the trellis moves the trajectory one stage to the right.

To avoid crowding in Fig. 5, branch labels appear at the left and right of the trellis rather than on the branch itself. For each state, the top label belongs to the top branch emanating from or incident to that state. Figure 6 uses thick arrows to show the same path emphasized in the state diagram of Fig. 3. However, in Fig. 3 the beginning and end of the path were not clear. In Fig. 6 the path clearly begins in state 00 and then travels through 01 and then 10 before returning to 00.

3.2. The Basic Viterbi Algorithm

The Viterbi algorithm uses the trellis diagram to compute the accumulated distances (called the *path metrics*) from the received sequence to the possible transmitted sequences. The total number of such trellis paths grows exponentially with the number of stages in the trellis, causing potential complexity and memory problems. However, the Viterbi algorithm takes advantage of the fact that the number of paths truly in contention to have the minimum distance is limited to the number of states in a single column of the trellis, assuming that ties may be arbitrarily resolved.

As an example of the Viterbi algorithm, consider transmission over the binary symmetric channel (bit error channel) where the probability of a bit error is less than $\frac{1}{2}$. On such a channel, maximum likelihood decoding reduces to finding the output sequence that differs in the fewest bit positions (has the minimum Hamming distance) from the received sequence. For this example, assume the encoder of Fig. 1 with the state diagram of Fig. 3 and the trellis of Fig. 5. For simplicity, assume that the receiver knows that the encoder begins in state 00.

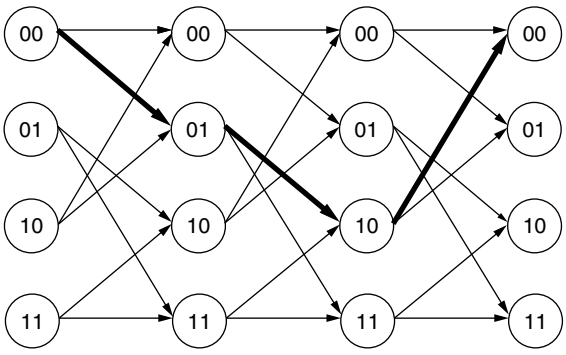

Figure 6. Trellis diagram for the path emphasized in Fig. 3.

Figure 7 illustrates the basic Viterbi algorithm for the received sequence 01 01 10. Beginning at the far left column, the only active state is 00. The circle representing this state contains a path metric of zero, indicating that as yet, the received sequence differs from the possible output sequences in no bit positions. Follow the two branches leaving the first column to the active states in the second column of the trellis. Branch metrics label each branch, indicating the Hamming distance between the received symbol and the symbol transmitted by the encoder when traversing that branch. The two hypothetical transmitted symbols are 00 for the top branch and 11 for the bottom (see Fig. 5). Since both differ in exactly one bit position from the received symbol 01, both branch labels are one.

The path metric for each destination state is the sum of the branch metric for the incident branch and the path metric at the root of the incident branch. In the second column, both path metrics are one since the root path metric is zero. These equal path metrics indicate that no path is favored at this point. Now follow the branches from the second column to the third. Exactly one branch reaches each state in the third column. Once again, adding the branch metric and the associated root path metric produces the new path metric.

When following branches from the third column to the fourth, two branches are incident on each state. Only the path with the minimum path metric needs to survive. For example, state 00 (the top state) in the fourth column has a path incident from state 00 in the third column with a path metric of $2 + 1 = 3$. It also has a path incident from state 10 in the third column with a path metric of $3 + 1 = 4$. Only the path with the smaller path metric needs to survive. Figure 7 shows the incident branches of survivor paths with thicker arrows than nonsurvivor paths. Each state in the fourth column has exactly one survivor path, and the values shown indicate the path metrics of the survivor paths.

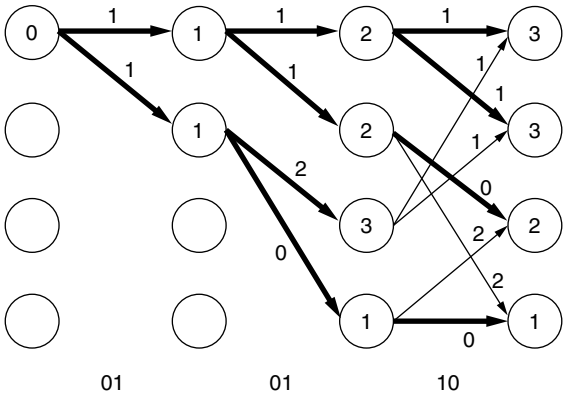

Figure 7. Illustration of the basic Viterbi algorithm on the bit error channel. This is also the trellis for hard Viterbi decoding on the AWGN channel in contrast to the soft Viterbi decoding shown in Fig. 8.

After all received symbols have been processed, the final step in decoding is to examine the last column and find the state containing the smallest path metric. This is state 11, the bottom state, in the fourth column. Following the survivor branches backward from the minimum-path-metric state identifies the trellis path of the maximum likelihood sequence. Reference to Fig. 5 reveals that the maximum likelihood path is the state trajectory $00 \rightarrow 01 \rightarrow 11 \rightarrow 11$. This state trajectory produces the output symbol sequence 11 01 10, which differs in exactly one bit position from the received sequence as indicated by its path metric. The input information sequence is decoded to be 1 1 1.

In this short example, only one trellis stage required path selection. However, once all states are active, path selection occurs with each trellis stage. In fact, if the initial encoder state is not known, path selection occurs even at the very first trellis stage. The basic computational module of the Viterbi algorithm is an add-compare-select (ACS) module. Adding the root path metric and incident branch metric produces the new path metric. Comparing the contending path metrics allows the decoder to select the one with minimum distance. When there is a tie (where two incident paths have the same path metric), a surviving path may be arbitrarily selected. In practice, ties are not uncommon. However, ties usually occur between paths that are ultimately destined to be losers.

3.3. Hard versus Soft Decoding

The integer branch and path metrics of the binary error channel facilitate a relatively simple example of the Viterbi algorithm. However, the AWGN channel is far more common than the bit error channel. For the AWGN channel, binary phase shift keying (BPSK) represents binary 1 with 1.0 and binary 0 with -1.0. These two transmitted values are distorted by additive Gaussian noise, so that the received values will typically be neither 1.0 nor -1.0. A novice might choose to simply quantize each received value to the closest of 1.0 and -1.0 and assign the appropriate binary value. This quantization would effectively transform the AWGN channel to the bit error channel, facilitating Viterbi decoding exactly as described above. This method of decoding is called hard decoding, because the receiver makes a binary (hard) decision about each bit before Viterbi decoding.

Hard decoding performs worse by about 2 dB than a more precise form of Viterbi decoding known as *soft decoding*. Soft decoding passes the actual received values to the Viterbi decoder. These actual values are called soft values because hard decisions (binary decisions) have not been made prior to Viterbi decoding. Soft Viterbi decoding is very similar to hard decoding, but branch and path metrics use squared Euclidean distance rather than Hamming distance. Figure 8 works an example analogous to that of Fig. 7 for the case where 1.0 and -1.0 are transmitted over the AWGN channel and soft Viterbi decoding is employed. A fixed-point implementation with only a few bits of precision captures almost all the benefit of soft decoding.

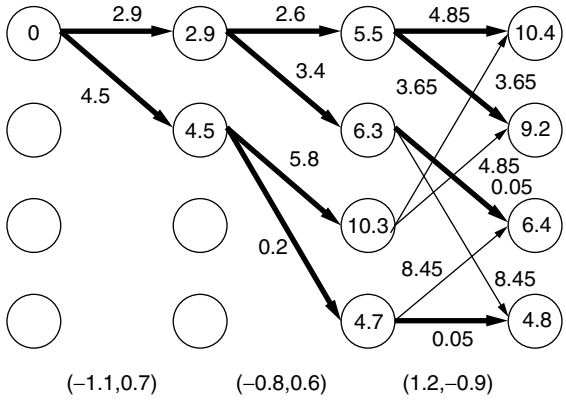

Figure 8. Illustration of soft Viterbi decoding.

3.4. Finite Traceback Viterbi

The maximum likelihood version of Viterbi decoding processes the entire received sequence and then selects the most likely path. Applications such as speech transmission can conveniently process relatively short data packets in this manner. However, stream-oriented applications such as a modem connection cannot wait until the end of the received sequence before making any decisions about the information sequence. In such cases, a suboptimal form of Viterbi decoding is implemented in which decisions are made about transmitted bits after a fixed delay. This fixed delay is called the traceback depth or decision depth of the Viterbi decoder. The exact choice of the traceback depth is usually determined by simulation, but there is an analytic technique that identifies a good lower bound on what the traceback depth should be. We will discuss this "analytic traceback depth" in the next section, since its computation is a natural by-product of computing the free distance of a convolutional code.

Figures 9 and 10 illustrate finite traceback Viterbi decoding. Figure 9 shows the path metrics and survivor paths (indicated by thick arrows) for a soft Viterbi decoder in steady state. Actually, the selection of survivor paths and path metrics is the same for maximum-likelihood Viterbi decoding and finite-traceback Viterbi decoding. Figure 10 shows the distinguishing behavior of finite-traceback Viterbi decoding. Rather than wait until the end of the received sequence, each k-bit input symbol is decoded after a fixed delay. In Fig. 10 this delay is three symbols. After each update of the path metrics, the path with the smallest metric (identified in Fig. 10 by a thick circle) is traced back three branches and the k-bit input symbol associated with the oldest branch is decoded. Notice that the paths selected by this algorithm do not have to be consistent with each other. For example, the two paths traced back in Fig. 10 could not both be correct, but this inconsistency does not force the decoded bits to be incorrect.

4. FREE DISTANCE

The ultimate measure of a convolutional code's performance is the bit error rate (BER) or block error rate

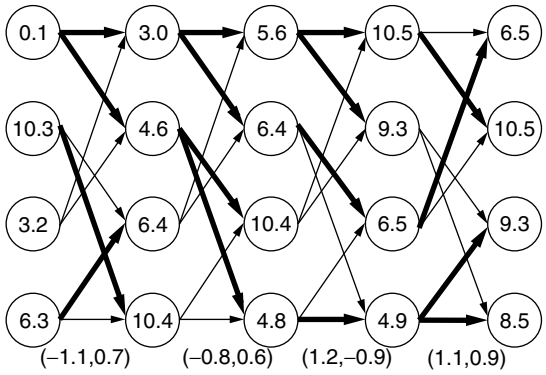

Figure 9. Steady state soft Viterbi state updates. Survivor paths are shown as thick arrows.

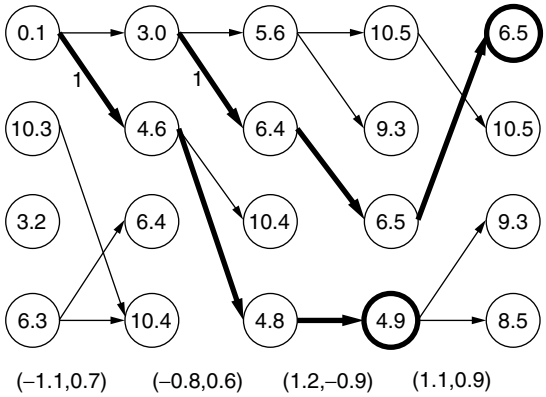

Figure 10. Finite traceback soft Viterbi decoding with a traceback depth of 3. Only the survivor paths of Fig. 9 are shown. Each traceback operation decodes only $k = 1$ bit. Thick arrows identify two such traceback paths.

(BLER) of the code as a function of signal-to-noise ratio (SNR). However, free distance gives a good indication of convolutional code performance. The free distance of a convolutional code is the minimum distance (either Hamming or Euclidean) between two distinct valid output sequences. Unlike algebraic block codes, which are designed to have specific distance properties, good convolutional codes are identified by an exhaustive search over encoders with a given number of memory elements. Often free distance is the metric used in these searches.

For simplicity, we will restrict the discussion of free distance to free Hamming distance. For BPSK, this restriction imposes no loss of generality since the Hamming and squared Euclidean free distances are related by a constant.

4.1. Computation of Free Distance

The set of distances from a codeword to each of its neighbors is the same for all codewords. Hence, the free distance is the distance from the all-zeros output sequence to its nearest-neighboring codeword. A Viterbi decoding operation with some special restrictions efficiently performs this computation. Viterbi decoding is performed on the undistorted all-zeros received sequence, but the first trellis branch associated with the correct path is disallowed. Thus prevented from decoding the correct sequence, the Viterbi algorithm identifies the nearest-neighbor sequence. Since the received sequence is noiseless, the path metric associated with the decoded sequence is the distance between that sequence and the all-zeros sequence, which is the free distance.

Figure 11 illustrates the computation of free Hamming distance using the Viterbi algorithm for the encoder described in Figs. 1, 3, and 5. The disallowed branch is shown as a dashed line. Only survivor branches are shown, and the thick branches indicate the minimum distance survivor path. Below each column is the minimum survivor path metric, which is called the *column distance*. The *free distance* is formally defined as the limit of the column distance sequence as the survivor pathlength tends to infinity. This limit is 5 in Fig. 11.

For noncatastrophic feedforward convolutional encoders, the free distance is equivalent to the minimum distance of a path that returns to the zero state. In general, the minimum distance path need not be the shortest path. For encoders with more states than the simple example of Fig. 11, there are typically several such paths having

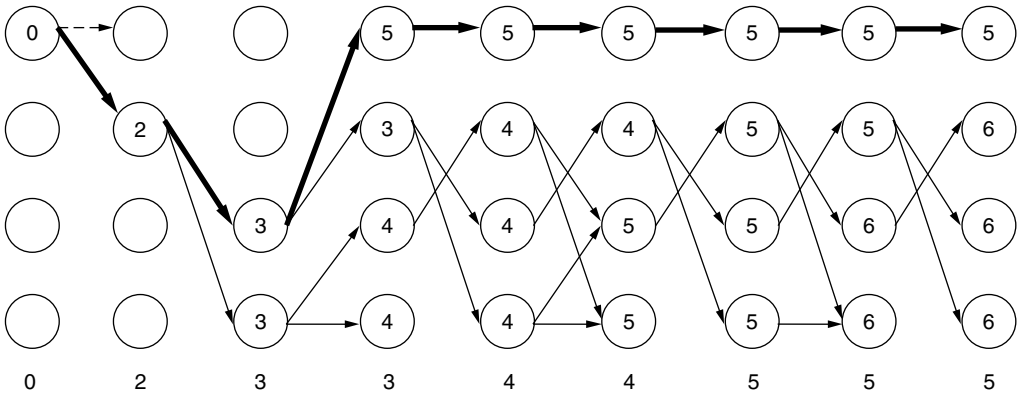

Figure 11. Application of the Viterbi algorithm to identify the free Hamming distance of the code described by Figs. 1, 3, and 5. The column distances are shown below each column. The disallowed branch is shown as a dashed line. Only survivor paths are shown, and the minimum distance path is shown with thick arrows.

the same minimum distance. The number of minimum-distance paths is the number of nearest-neighbor output sequences. This is sometimes called the *multiplicity* of the free distance. If two codes have the same free distance, the code with the smaller multiplicity is preferred.

4.2. Analytic Decision Depth

As mentioned in Section 3.4, the specific decision depth used in finite traceback Viterbi is usually determined by simulation. However, a good estimate of the decision depth helps designers know where to begin simulations. For noncatastrophic feedforward encoders, Anderson and Balachandran [16] compute a useful lower bound on the required decision depth as a by-product of the free-distance computation.

This analytic decision depth is the pathlength at which the survivor path incident on the zero state has a path metric that is the unique minimum distance in the column. In other words, the path metric of the survivor path to the zero state is the only distance in the column equal to the column distance. In Fig. 11, this analytic decision depth is 8; after the eighth branch the path metric of the survivor path to the zero state is the only distance in the column equal to the column distance of 5. For noncatastrophic feedforward encoders, the Viterbi decoding procedure for computing free distance may be stopped when the analytic decision depth is identified. The column distance remains fixed thereafter.

When using this analytic decision depth, finite traceback decoding gives performance consistent with the first-order metrics of free distance and multiplicity. The asymptotic performance (in the limit of high SNR) is the same as maximum-likelihood Viterbi. In practice, a somewhat larger decision depth is often used to capture some additional performance at SNRs of interest by improving second-order metrics of performance (i.e., distances slightly larger than the minimum distance). For example, the analytic decision depth of the standard rate-$\frac{1}{2}$ 64-state feedforward convolutional encoder is 28, but simulation results show that a decision depth of 35 gives a noticeable performance improvement over 28. Decision depths larger than 35 give only negligible improvement.

4.3. Catastrophic Encoders

A convolutional encoder is catastrophic if a finite number of errors in the output sequence can cause an infinite number of errors in the input sequence. With such an encoder, the consequences of a decoding error can be truly catastrophic. Catastrophic encoders are certainly undesirable, and they are never used in practice. In fact, they are easily avoided because they are not minimal encoders. Hence if an encoder is catastrophic it also uses more memory elements than does a minimal equivalent encoder, which is not catastrophic.

An encoder is catastrophic if and only if its state diagram has a loop with zero output weight and nonzero input weight. Catastrophic encoders still have a free distance as defined by the limit of the column distance, but this free distance is seldom equal to the minimum survivor path metric to the zero state. Usually, some of the survivor path metrics for nonzero states never rise above the minimum survivor path metric to the zero state. An additional stopping rule for the Viterbi decoding computation of free distance resolves this problem: If the column distance does not change for a number of updates equal to the number of states, the free distance is equal to that column distance.

Noncatastrophic encoders may also require this additional stopping rule if they have a nontrivial zero-output-weight loop. Such a loop does not force catastrophic behavior if it is also a zero-input-weight loop. Such a situation only occurs with feedback encoders since feedforward encoders do not have loops with zero output weight and zero input weight except the trivial zero-state self-loop. In cases where this stopping criterion is required, the analytic decision depth of Anderson and Balachandran is not well defined. However, a practical place to start simulating decision depths is the pathlength at which the Viterbi computation of free distance terminates.

Because nontrivial zero-output-weight lops indicate a nonminimal encoder, their free distance is not often computed. However, there are circumstances where computation of the free distance is still interesting. As described by Fragouli et al. [17], these “encoders” arise not from poor design but indirectly when severe erasures in the channel transform a minimal encoder into a weaker, nonminimal encoder. An alternative to the additional stopping rule is simply to compute the free distance of an equivalent minimal encoder.

5. BOUNDS ON BIT ERROR RATE

As mentioned at the beginning of Section 4, BER and BLER as functions of SNR are the ultimate metrics of convolutional code performance. Monte Carlo simulation plays an important role in the characterizing BER and BLER performance. However, accurate characterization by Monte Carlo simulation at very low BER or BLER, say, less than 10^{-10}, is not computationally feasible with today’s technology. However, analytic upper bounds on BER are very accurate below BER 10^{-5}. Thus, the use of bounds in conjunction with Monte Carlo simulation for high BER provides a good overall performance characterization.

5.1. The Generating Function

To facilitate the bound, a generating function or transfer function enumerates in a single closed-form expression all paths (including nonsurvivors) that return to the zero state in an infinite extension of the trellis of Fig. 11. The bound itself is analogous to the moment generating function technique for computing expectations of random variables. Figure 12 shows an altered version of the state diagram of Fig. 3 where the zero state has been split into a beginning zero state and an ending zero state. This new diagram is called a *split-state diagram*. For the bound, only the Hamming weights of input and output symbols are needed. These Hamming weights are given as exponents of I and W, respectively. These values appear as exponents, so that when the labels along any

path from the beginning zero state to the ending zero state are multiplied, the result is a single expression $I^i W^w$, where i is the overall input Hamming weight of the path and w is the overall output Hamming weight of the path.

Let A be the matrix of branch labels for all branches that neither begin nor end in a zero state. Each column of A represents a beginning state, and each row represents an ending state for a branch. A zero indicates no branch between the corresponding beginning and ending states. For Fig. 12

$$A = \begin{bmatrix} 0 & I & 0 \\ W & 0 & W \\ IW & 0 & IW \end{bmatrix} \tag{1}$$

Let b be the column of branch labels for all branches that begin in the zero state. For Fig. 12

$$b = \begin{bmatrix} IW^2 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

Let c be the row of branch labels for all branches that end in the zero state. For Fig. 12

$$c = \begin{bmatrix} 0 & W^2 & 0 \end{bmatrix} \tag{3}$$

The shortest path from the beginning zero state to the ending zero state has three branches; it is the path shown by thick arrows in Fig. 11. The product of the labels for this path may be computed as $cAb = IW^5$. Note that the exponent of 5 is consistent with the path metric of 5 in Fig. 11, which is also the free distance. There is one four-branch path and its label product is $cA^2b = I^2W^6$. There are two five-branch paths and their label products are $cA^3b = I^3W^7 + I^2W^6$. In general, $cA^{L-2}b$ gives the label products of all L-branch paths. Thus, the equation

$$T(W, I) = \sum_{L=3}^{\infty} cA^{L-2}b \tag{4}$$

$$= c(I - A)^{-1}b \tag{5}$$

enumerates the label products of all paths from the beginning zero state to the ending zero state. Note that I is the input weight indeterminate in Eq. (4) but the identity

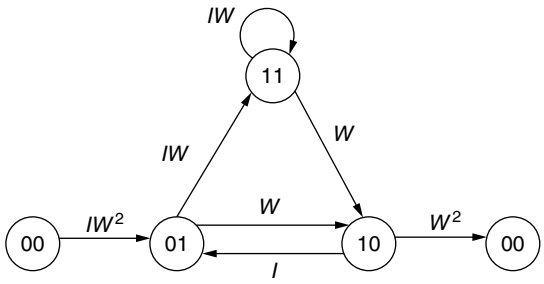

Figure 12. Split-state diagram for the encoder described by Figs. 1, 3, and 5. The exponent of W indicates the Hamming weight of the output error symbol. The exponent of I indicates the Hamming weight of the input error symbol.

matrix in Eq. (5). $T(W, I)$ is the generating function (or transfer function) of the convolutional encoder.

5.2. Union Bounds

Manipulation of $T(W, I)$ produces upper bounds on the BER for both the bit error channel and the AWGN channel. These upper bounds compute the sum over all error events e

$$\sum_e i_e P_e \tag{6}$$

where an error event is simply a path from the beginning zero state to the ending zero state. The input Hamming weight i_e associated with the error event e counts the total number of bit errors that all shifts of this error event can induce on a fixed symbol position. P_e is the probability that this path is closer to the received sequence than the transmitted (all-zeros) sequence. Note that (6) is an upper bound because P_e does not subtract probability for situations where more than one path is closer to the received sequence than the all-zeros sequence. For the bit error channel with bit error probability p,

$$\text{BER} \le \frac{1}{k} \left\{ \frac{\partial T(W, I)}{\partial I} \right\}_{I=1, W=2(p-p^2)^{1/2}} \tag{7}$$

For BPSK transmission of $\pm E_s^{1/2}$ over the AWGN channel with noise variance $N_0/2$, we obtain

$$\text{BER} \le \frac{1}{k} Q\left[\left(\frac{2d_{\text{free}} E_s}{N_0} \right)^2 \right] e^{d_{\text{free}} E_s / N_0} \times \left\{ \frac{\partial T(W, I)}{\partial I} \right\}_{I=1, W=e^{-E_s/N_0}} \tag{8}$$

$$\le \frac{1}{2k} \left\{ \frac{\partial T(W, I)}{\partial I} \right\}_{I=1, W=e^{-E_s/N_0}} \tag{9}$$

where the tighter bound of Eq. (8) requires knowledge of the free distance d_{free}, but the looser bound of Eq. (9) does not.

6. FINAL REMARKS

Although a block code has a well-defined blocklength, convolutional codes do not. Convolutional codes are sometimes considered to have infinite blocklength, and this perspective is valuable for certain derivations, such as the derivation of union bounds on BER presented in Section 5. However, Sections 3.4 and 4.2 demonstrate that most of the useful information for decoding a particular input symbol lies within a relatively small interval of output symbols called the *decision depth*. In two important senses of blocklength, the latency required for decoding and the general strength of the code, the (properly chosen) decision depth is a good indicator the effective blocklength of a convolutional code. The decision depth of standard convolutional codes is small, certainly less than 50 for the standard rate-$\frac{1}{2}$ code with six memory elements in a single shift register.

Shannon's channel capacity theorem [18] (see also the treatise by Cover and Thomas [19]) computes the maximum rate that can be sent over a channel (or the maximum distortion that can be tolerated for a given rate). This theorem applies only as blocklength becomes infinite; in general it is not possible to achieve the performance promised by Shannon with small-blocklength codes. Indeed, convolutional code performance is hampered by their relatively small effective blocklength. For a bit error rate (BER) of 10^{-5}, they typically require about 4 dB of additional signal-to-noise ratio (SNR) beyond the Shannon requirement for error free transmission in the presence of AWGN. In contrast, for a BER of 10^{-5} Turbo codes and low-density parity-check codes, which both typically have blocklengths on the order of 10^3 or 10^4, require less than 1 dB of additional SNR beyond the Shannon requirement for error-free transmission in AWGN.

On the other hand, the performance of convolutional codes is actually quite good, given their short blocklengths. Applications such as speech transmission that require very low latency continue to employ convolutional codes because they provide excellent performance for their low latency and may be decoded with relatively low complexity. Furthermore, since Turbo codes contain convolutional encoders as constituents, a good understanding of convolutional codes remains essential even for long-blocklength applications.

BIOGRAPHY

Richard D. Wesel received both B.S. and M.S. degrees in electrical engineering from MIT in 1989 and the Ph.D. degree in Electrical Engineering from Stanford University in 1996. From 1989 to 1991 he was with AT&T Bell Laboratories, where he worked on nonintrusive measurement and adaptive correction of analog impairments in AT&T's long-distance network and the compression of facsimile transmissions in packet-switched networks. He holds patents resulting from his work in both these areas.

From July 1996 to July 2002 he was an Assistant Professor in the Electrical Engineering Department of the University of California, Los Angeles. Since July 2002 he is an Associate Professor at UCLA. His research is in communication theory with particular interests in the topics of channel coding the distributed communication. In 1998 he was awarded a National Science Foundation CAREER Award to pursue research on robust and rate-compatible coded modulation. He received an Okawa Foundation Award in 1999 for research in information and telecommunications, and he received the 2000 TRW Excellence in Teaching Award from the UCLA School of Engineering and Applied Science. Since 1999 he has been an Association Editor for the *IEEE Transactions on Communications* in the area of coding and coded modulation.

BIBLIOGRAPHY

1. P. Elias, Coding for noisy channels, *Proc. IRE Conv. Rec. part 4* 37–46 (1955) (this paper is also available in Ref. 2).
2. E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, IEEE Press, 1974.
3. S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.
4. R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, 1999.
5. G. D. Forney, Jr., Convolutional codes I: Algebraic structure, *IEEE Trans. Inform. Theory* **16**(6): 720–738 (Nov. 1970).
6. R. Johannesson and Z. Wan, A linear algebra approach to minimal convolutional encoders, *IEEE Trans. Inform. Theory* **39**(4): 1219–1233 (July 1993).
7. J. M. Wozencraft, Sequential decoding for reliable communication, *Proc. IRE Conv. Rec. part 2* 11–25 (1957).
8. R. M. Fano, A heuristic discussion of probabilistic decoding, *IEEE Trans. Inform. Theory* **9**: 64–74 (April 1963).
9. K. Sh. Zigangirov, Some sequential decoding procedures, *Probl. Peredachi Inform.* **2**: 13–25 (1966) (in Russian).
10. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimal decoding algorithm, *IEEE Trans. Inform. Theory* **13**: 260–269 (April 1967).
11. G. D. Forney, Jr., The Viterbi algorithm, *Proce. IEEE* **61**: 268–278 (March 1973).
12. G. D. Forney, Jr., Convolutional codes II: Maximum likelihood decoding, *Inform. Control* **25**: 222–266 (July 1974).
13. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, *IEEE Trans. Inform. Theory* **20**(2): 248–287 (March 1974).
14. C. Berrou, A. Glavieux, and P. Thitimajshima, Near Shannon limit error correcting coding and decoding: Turbo-codes, *Proc. Int. Conf. Communication*, May 1993, pp. 1064–1070.
15. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, A soft-input soft-output APP module for the iterative decoding of concatenated codes, *IEEE Commun. Lett.* **1**(1): 22–24 (Jan. 1997).
16. J. B. Anderson and K. Balachandran, Decision depths of convolutional codes, *IEEE Trans. Inform. Theory* **35**(2): 455–459 (March 1989).
17. C. Fragouli, C. Komninakis, and R. D. Wesel, Minimality under periodic puncturing, *IEEE Int. Conf. Communication*, Helsinki, Finland, June 2001, pp. 300–304.
18. C. E. Shannon, A mathematical theory of communication, *Bell Syst. Tech. J.* **27**: 379–423, 623–656 (1948).
19. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, 1991.