OFFPRINT FROM

CODING AND COMPLEXITY

edited by

G. LONGO
University of Trieste, Italy

CISM COURSES AND LECTURES No. 216
International Centre for Mechanical Sciences
SPRINGER–VERLAG WIEN NEW YORK, 1976

ERROR BOUNDS FOR TREE CODES, TRELLIS CODES, AND CONVOLUTIONAL CODES WITH ENCODING AND DECODING PROCEDURES *

James L. Massey
Freimann Professor of Electrical Engineering
University of Notre Dame
Notre Dame, Indiana U.S.A.

* The original work reported herein was supported in part by the U.S.A. National Aeronautics and Space Administration under NASA Grant NGL 15-004-026 at the University of Notre Dame in liaison with the Goddard Space Flight Center.

1. Introduction

One of the anomalies of coding theory has been that while block parity-check codes form the subject for the overwhelming majority of theoretical studies, convolutional codes have been used in the majority of practical applications of "error-correcting codes." There are many reasons for this, not the least being the elegant algebraic characterizations that have been formulated for block codes. But while we may for aesthetic reasons prefer to speculate about linear block codes rather than convolutional codes, it seems to me that an information-theorist can no longer be inculpably ignorant of non-block codes. It is the purpose of these lectures to provide a reasonably complete and self-contained treatment of non-block codes for a reader having some general familiarity with block codes.

In order to make our presentation cohesive, we have borrowed heavily from other authors (hopefully with appropriate credit) and have included some of our own work that was previously unpublished. In the latter category are the formulations of the general ensembles of tree codes and trellis codes, given in Sections 3 and 4 respectively, together with the random coding bound for the ensemble of tree codes in Section 3. The random coding bound of Section 4 for trellis codes was, of course, given earlier by Viterbi for convolutional codes (which are the linear special case of trellis codes as we define the latter) for the case $T = M$ but we have somewhat generalized this bound.

Also in the category of our previously unpublished work is the low "complexity" ensemble of convolutional codes which is shown in Section 7 to meet the same upper bound on decoding error probability as for the entire ensemble of convolutional codes. There are a number of other places in our treatment where the reader familiar with past work in convolutional coding will recognize some novel treatments of the subject matter and we hope that these have contributed to the cohesiveness of these lectures.

2. The Two-Codeword-Exponent for Discrete Memoryless Channels

The discrete memoryless channel (or DMC) is a channel with a finite input alphabet and a finite output alphabet that acts independently on each input digit and whose statistics do not vary with time. Letting $A = \{a_1, a_2, \ldots, a_q\}$ and

$B = \{b_1, b_2, \ldots b_{q'}\}$ be the input and output alphabets respectively, we can specify a DMC by stating the conditional probability $P(b_j \mid a_i)$ of receiving b_j when a_i is transmitted for $j = 1,2,\ldots q'$ and $i = 1,2,\ldots q$. A DMC is often shown by a directed graph in which the edge from node a_i to node b_j is labelled with $P(b_j \mid a_i)$ as shown in Fig. 2.1 for the binary symmetric channel (BSC) for which $A = B = \{0,1\}$. The quantity ϵ is called the "crossover probability" of the BSC.

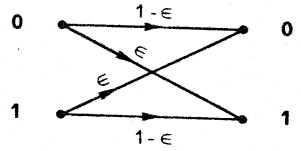

Fig. 2.1 The Binary Symmetric Channel (BSC)

An (N,R) **block code** for a DMC is an ordered set of $m = 2^{NR}$ N-tuples over the input alphabet A of the DMC, $(\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m)$. We shall write $\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots x_{jN}]$. The parameters N and R are the code length and rate respectively. We say that R is the rate in "bits per channel use" because, when the m codewords are equally likely, we send $\log m = NR$ bits of information in N uses of the DMC. (Here and hereafter, the base 2 is used for all logarithms.)

A maximum likelihood decoder (MLD), when $\mathbf{y} = [y_1, y_2, \ldots y_N]$ is received over the DMC, chooses as its estimate of the index of the transmitted codeword (one of) the index (es) j which maximizes

$$P(\mathbf{y} \mid \mathbf{x}_j) = \prod_{n=1}^{N} P(y_n \mid x_{jn}).$$

Consider now the simplest interesting codes, i.e. codes with only $m = 2$ codewords or $(N, R = 1/N)$ codes. Let Q be a probability distribution over the channel input alphabet A. To each code $(\mathbf{x}_1, \mathbf{x}_2)$, we assign the probability

$$P(\mathbf{x}_1, \mathbf{x}_2) = \prod_{n=1}^{N} Q(x_{1n})\, Q(x_{2n})$$

which is the probability of selecting that code by choosing each code digit independently according to Q. Let $P_e(\mathbf{x}_1, \mathbf{x}_2)$ be the decoding error probability with a MLD for the code $(\mathbf{x}_1, \mathbf{x}_2)$ and a given probability assignment on the codewords. Then

$$\bar{P}_e = \sum_{\mathbf{x}_1 \epsilon A^N} \sum_{\mathbf{x}_2 \epsilon A^N} P_e(\mathbf{x}_1, \mathbf{x}_2)\; P(\mathbf{x}_1, \mathbf{x}_2)$$

is the average error probability with MLD for the ensemble of codes with $m = 2$

codewords of length N.

It is not difficult to show (Cf. Gallager [1]) that, regardless of the probability assignment on the two codewords,

$$\bar{P}_e \leq 2^{-NR_o} \tag{2.1}$$

where

$$R_o = -\log\{\min_Q \sum_{y\epsilon B} [\sum_{x\epsilon A} Q(x)\sqrt{P(y|x)}]^2\} \tag{2.2}$$

when Q is taken as the minimizing distribution in (2.2). Since there must be at least one code whose $P_e(x_1,x_2)$ is no worse than average, it follows from (2.1) that we can find, for increasing N, a sequence of codes whose decoding error probability with MLD remains below the decreasing exponential 2^{-NR_o}. The exponent R_o, as defined by (2.2), is called the **two-codeword-exponent** for the DMC.

For DMC's with a binary input alphabet A = {0,1 }, the calculation of R_o is simplified by the fact that Q(0) = Q(1) = 1/2 is always the minimizing distribution in (2.2). For instance, for the BSC one easily finds that

$$R_o \text{ (for BSC)} = 1\text{-}\log[1 + 2\sqrt{\epsilon(1-\epsilon)}]. \tag{2.3}$$

In particular, we find from (2.3) that R_o is .45, .50 and .55 when ϵ is .057, .045 and .033 respectively.

In the following sections, we shall use R_o to find bounds on $\bar{P}_e$ with MLD for much more interesting code ensembles than that of codes with two codewords.

3. Tree Codes

We define an (L,T) m-ary tree to be a rooted tree such that (1) m branches diverge from each node at depth less than L from the root node and (2) one branch diverges from each node at depth less than L +T but at least L from the root. Here, L, T and m are integers such that $L \geqslant 1$, $T \geqslant 0$ and $m \geqslant 2$. We call L and T the **dividing length** and **tail length** respectively of the tree. In Fig. 3.1, we show the (L = 3, T = 2) binary tree. We note in general that the (L,T) m-ary tree has m^L terminal nodes and these terminal nodes are at depth L + T from the root.

We now define an (N,R,L,T) **tree code** for a DMC as the assignment of N channel input letters to each branch of an (L,T) (2^{NR})-ary tree. Note that $m = 2^{NR}$ is the number of branches diverging from each node in the "dividing part" of the

tree. We call R the rate of the tree code and we define the **constraint length** of the tree code as

$$N_t = (T+1)N \qquad (3.1)$$

which is just the number of digits on the "long paths" from the last dividing nodes to the terminal nodes.

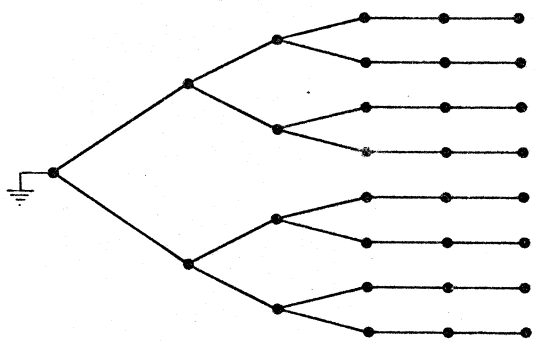

Fig. 3.1 The (L=3, T=2) Binary Tree

An (N,R,L,T) tree code is a special type of block code for the DMC. As a block code, it has $m_B = m^L$ codewords of length $N_B = (L+T)N$, namely the m_B N_B-tuples which label the paths from the root to each terminal node. As a block code, its rate R_B is given by

$$R_B = \log(m_B)/N_B = \frac{L}{L+T} R \qquad (3.2)$$

which, for the usual case in practice where $L \gg T$, is approximately the same as its rate R as a tree code.

We now derive an upper bound on the average decoding error probability, $\overline{P}_e$, for MLD of the ensemble of (N,R,L,T) tree codes for a given DMC when each code is assigned the probability equal to selecting that code when each channel input digit placed in the (L,T) m-ary tree is selected independently according to the minimizing distribution Q in (2.2).

Let E_i, $1 \leqslant i \leqslant L$, be the event that some path to a terminal node diverging from the correct path at the node at depth L-i from the root is at least as likely to produce the received sequence as the corresponding segment of the correct path. Then

$$P_e \leqslant P(E_1 \cup E_2 \cup \ldots \cup E_L)$$

where the inequality is needed because we might decode correctly when the correct path is "tied" with another path as most probable. By the union bound,

$$P_e \leqslant P(E_1) + P(E_2) + \ldots P(E_L). \qquad (3.3)$$

There are $(m-1)m^{i-1}$ paths to terminal nodes which diverge from the correct path from the node at depth L-i from the root and each of these paths is T+i branches,

or $(T+i)N$ channel input digits in length. The probability of one of these paths being more likely than the corresponding segment of the correct path is just the error probability for two codewords of length $(T+i)N$. Thus, using (2.1) and a union bound, we have

$$\overline{P(E_i)} \leqslant (m-1)m^{i-1}\, 2^{-(T+i)NR_o}. \tag{3.4}$$

Now, averaging in (3.3) and making use of (3.4), we obtain

$$\overline{P}_e \leqslant (m-1)2^{-(T+1)NR_o} \sum_{i=1}^{L} m^{i-1}\, 2^{-(i-1)NR_o}.$$

We recognize the summation above as a geometric series which, when $m < 2^{NR_o}$, converges upon letting $L \to \infty$ to give

$$\overline{P}_e \leqslant \frac{m-1}{1-m\,2^{-NR_o}}\, 2^{-(T+1)NR_o},\ m < 2^{NR_o}.$$

Finally, overbounding $m-1$ by m in the numerator and using $m = 2^{NR}$ and $N_t = (T+1)N$, we obtain the result that the average decoding error probability for the ensemble of (N,R,L,T) tree codes with MLD on a DMC satisfies

$$\overline{P}_e < c_t\, 2^{-N_t R_o} \text{ when } R < R_o \tag{3.5}$$

where

$$c_t = \frac{1}{2^{-NR} - 2^{-NR_o}} \tag{3.6}$$

is a relatively unimportant constant for any fixed rate R, $R < R_o$.

The bound (3.5) is quite remarkable in several respects. First, it shows that we can operate with tree codes at rates very close to R_o and still obtain R_o as the exponent of error probability just as if there were only two codewords. There is no need to use small information rates R to get a rapid decrease of error probability with constraint length. Second, the bound is completely independent of the dividing length L of the tree and depends only on the tail length T through $N_t = (T+1)N$.

Thus we can take $L \gg T$ to ensure, by (3.2), that the "nominal" rate R of the tree code is very close to the true information rate in bits per channel use. Third and finally, our derivation of the bound was extremely simple.

It is possible to extend the arguments used here to obtain a bound on $\bar{P}_e$ for tree codes in the region $R_0 \leqslant R < C$ where C is channel capacity for the DMC. For details of this argument, the reader is referred to Johannesson [2]. The case of greatest practical interest, however, is for R slightly less than R_0 for which we have no need of the more general bound.

We have already noted that an (N,R,L,T) tree code for a DMC is a special type of $(N_B = (L+T)N,\ R_B = RL/(L+T))$ block code. It is also interesting to note that we can consider an (N,R) block code for a DMC to be the special case of an $(N,R,\ L = 1,\ T = 0)$ tree code. Whether block codes or tree codes are more general is really just a matter of taste. To our taste, we prefer to think of the block code as the special case of a tree code.

4. Trellis Codes

By introducing "memory" into tree codes, we shall be led naturally to a very interesting class of codes which we call "trellis" codes for reasons that will become obvious.

Consider first how one might encode an (N,R,L,T) tree code. Since $m = 2^{NR}$ branches diverge from each node at depth less than L from the root, we can use a sequence $i_0, i_1, \ldots, i_{L-1}$ of L m-ary information digits (whose alphabet for the moment we shall consider to be $\{0,1,\ldots,m-1\}$) to select which branch to follow as we move away from the root. In Fig. 4.1, we show our rule for choosing the appropriate branch.

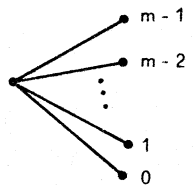

Fig. 4.1 The Convention by Which an Information Digit Selects a Branch to Follow

Thus, for instance, the information sequence $i_0, i_1, i_2 = 0,0,0$ would cause transmission of the channel input letters on the lowermost path of the tree as in Fig. 3.1 for an $(N, R = 1/N, L = 3, T = 2)$ tree code.

Suppose now for some integer M, $T \leqslant M \leqslant L + T$, we label each node in an (L,T) m-ary tree with the previous M information digits leading to that node. By way of convention, we take information digits previous to the origin as 0's and, similarly, we take 0 as the "information digit" corresponding to the unique path from each node at depth less than L+M but L or more from the root. We then

obtain what we shall call an (L,T,M) node-labelled m-ary tree. In Figure 4.2, we show the (L = 3, T = 2, M = 2) node-labelled binary tree.

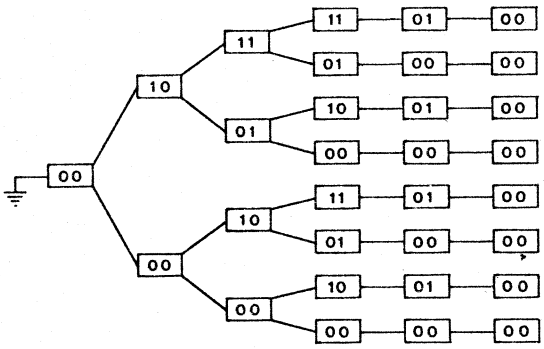

Fig. 4.2 The (L = 3, T = 2, M = 2) Node-Labelled Binary Tree

Suppose now that the tree code has "memory" M in the sense that for any two nodes at the same depth with the same label (i.e. having the same M previous information digits) the same remaining encoded sequence will result when the same sequence of information digits is applied beginning at either node. We then need to retain only one node with each label at each depth and all branches previously entering nodes with this label can be made to enter this single node with the label. We define an (L,T,M) m-ary trellis to be the result of such coalescing of similarly labelled nodes in an (L,T,M) node-labelled m-ary tree. In Figure 4.3, we show (a) the (L = 3, T = 2, M = 2) binary trellis and (b) the (L = 3, T = 1, M = 2) binary trellis. In general, the (L,T,M) trellis will have m^{M-T} terminal nodes.

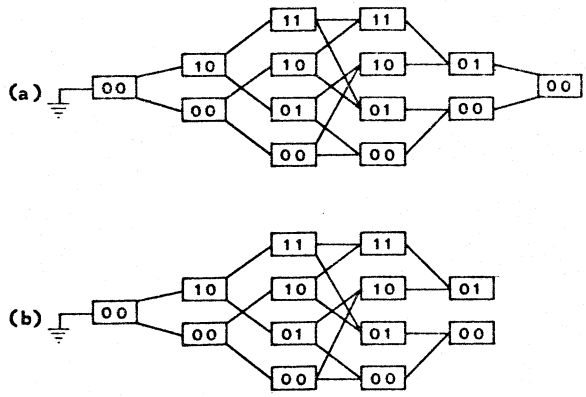

Fig. 4.3 (a) The (L = 3, T = 2, M = 2) Binary Trellis and (b) The (L = 3, T = 1, M = 2) Binary Trellis

We now define an (N,R,L,T,M) trellis code for a DMC as the assignment of N channel input letters to each branch of an (L,T,M) (2^{NR})-ary trellis.

As our discussion has made clear, an (N,R,L,T,M) trellis code for a DMC is a special type of (N,R,L,T) tree code so we still continue to call

$$N_t = (T + 1)N$$

the constraint length of the trellis code and, moreover, (3.2) still relates the "nominal" rate R of the trellis code to its true rate R_B in bits transmitted per use of

the channel.

We now proceed to derive an upper bound on the average error probability for MLD on a DMC over the ensemble of trellis codes when we assign to each code the probability of its selection when each channel input digit placed on the (L,T,M) m-ary trellis is selected independently according to the minimizing distribution Q in (2.2). We consider first the special case M = T from which we shall later infer the general result with very little additional effort. Note that, when M = T, there is just one terminal node in the trellis, namely the node labelled 00...0. Thus, every path through the trellis starts at the root and ends at this node. (See Fig. 4.3(a).) Again we let E_i be the event that some path to the terminal node which diverges at depth L - i from the correct path is at least as likely to produce the received sequence as the corresponding segment of the correct path. Then the bound (3.3) again applies for MLD, i.e.

$$P_e \leqslant \sum_{i=1}^{L} P(E_i) \tag{4.1}$$

Next, we further decompose the events E_i. A path diverging at depth L -i can first remerge with the correct path at depth L + M + 1 - i since its information digits must agree with those on the correct path for M successive branches for remergence to occur. Let A_{ij}, $1 \leqslant j \leqslant i$, be the event that some path diverging at depth L - i and first remerging at depth L + M + j - i is at least as likely to produce the received sequence as the corresponding M + j branch segment of the correct path. There are at most $m^{j-1}(m-1)$ such paths since M of their information digits agree with those on the correct path and their first information digit disagrees with that on the correct path. The probability of one of these paths being more likely than the correct path is just the error probability for two codewords of length (M + j)N. Thus, using (2.1) and a union bound, we have

$$\overline{P(A_{ij})} \leqslant (m-1)m^{j-1}\, 2^{-(M+j)NR_o}. \tag{4.2}$$

Moreover, we note that

$$E_i = A_{i1} \cup A_{i2} \cup \ldots \cup A_{ii}$$

so that a union bound gives

$$P(E_i) \leqslant \sum_{j=1}^{i} P(A_{ij}).$$

Taking averages and using (4.2), we obtain

$$\overline{P(E_i)} \leqslant (m-1)\, 2^{-(M+1)NR_o} \sum_{j=1}^{i} m^{j-1}\, 2^{-(j-1)R_o}.$$

Overbounding by letting $i \to \infty$, we recognize the above expression to be precisely the same as that leading to (3.5) with T replaced by M. Thus, we have

$$\overline{P(E_i)} < c_t\, 2^{-(M+1)NR_o},\ R < R_o \tag{4.3}$$

where c_t is defined in (3.6). Finally, taking averages in (4.1) and using (4.3), we obtain

$$\overline{P}_e \leqslant c_t\, L\, 2^{-(M+1)NR_o},\ R < R_o \tag{4.4}$$

as our basic bound for MLD of the ensemble of (N,R,L,T,M = T) trellis codes. Unlike the case for tree codes, this bound depends linearly on the dividing length L of the trellis and one can easily convince himself that this dependence on L is real and not merely the result of a weakness in our bounding arguments.

We now extend the above arguments to apply to the ensemble of (N,R,L,T,M) trellis codes without the restriction that M = T. We have already pointed out that the trellis will have m^{M-T} terminal nodes in the general case and that $T \leqslant M \leqslant L+T$.

Suppose that some path to a terminal node diverging from the correct path at depth L - i is more likely than the corresponding segment of the correct path. If this path remerges with the correct path, then such an error event is included in the event E_i for a trellis code with T = M. If this path does not remerge with the correct path, then such an error event is included in the event E_i for a tree code with tail length T. Thus $\overline{P}_e$ for MLD of the ensemble of (N,R,L,T,M) trellis codes is overbounded by the sum of $\overline{P}_e$ for the ensemble of (N,R,L,T = M,M) trellis codes and $\overline{P}_e$ for the ensemble of (N,R,L,T) tree codes, that is

$$\overline{P}_e < c_t L\, 2^{-(M+1)NR_o} + c_t\, 2^{-(T+1)NR_o}$$

which we may rewrite as

$$\bar{P}_e < c_t\,[1 + L2^{-(M-T)NR_o}]\,2^{-N_tR_o} \tag{4.5}$$

when $R < R_o$. Inequality (4.5) is our basic bound for MLD of the ensemble of (N,R,L,T,M) trellis codes.

Inequality (4.5) provides the clue for avoiding the undesirable dependence on L as in (4.4) for trellis codes with M = T. For suppose we choose

$$L2^{-(M-T)NR_o} \leqslant 1$$

or, equivalently

$$M - T \geqslant \frac{\log L}{NR_o}\ , \tag{4.6}$$

then (4.5) becomes

$$\bar{P}_e \leqslant 2\,c_t\,2^{-N_tR_o}. \tag{4.7}$$

We conclude that making M only slightly greater than T according to (4.6) is sufficient to reduce P_e for a trellis code very close to that for a full tree code.

We have already noted that an (N,R,L,T,M) trellis code for a DMC is a special type of (N,R,L,T) tree code. It is interesting to note that we can alternatively think of the (N,R,L,T) tree code as the special case of an (N,R,L,T,M = L +T) trellis code since, when M = L +T, the trellis degenerates to a tree with $m^{M-T} = m^L$ terminal nodes.

The term "trellis" was first coined by Forney to describe graphs of the type in Fig. 4.3(a) which he employed to study convolutional codes. As we shall see, convolutional codes are a special subclass of the class of trellis codes which we defined here. The bounds (4.5) and (4.7) have been extended to rates R in the region $R_o < R < C$ by Johannesson [2] who also reported simulation results showing that the necessary excess of memory over tail length to make P_e independent of L is very closely given by the righthand side of (4.6).

5. Tree and Trellis Encoders--Convolutional Encoders

The observation above that the ensemble of (N,R,L,T) tree codes for a DMC coincides with the ensemble of (N,R,L,T,M = L+ T) trellis codes permits us to consider encoders for trellis codes alone without loss of generality.

Proceeding as in the previous section, we suppose that $i_0, i_1, \ldots i_{L-1}$ are the m-ary "information digits" to be encoded, each of which controls the branch to be followed by the encoder at the nodes within the dividing length of the trellis. We shall associate a "time unit" with each encoded branch, and hence we may write the encoded sequence as $\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{L-1}, \mathbf{t}_L, \ldots, \mathbf{t}_{L+T-1}$ where $\mathbf{t}_u$, the encoded branch at time u, is an N-tuple of q-ary channel input digits. Similarly, $\mathbf{i}_u$ is the "information digit" at time u for $0 \leqslant u < L$. We suppose that the sequence $i_0, i_1, \ldots i_{L-1}$ is augmented by a "tail" of T 0's where 0 is just some designated digit in the m-ary alphabet. For convenience, we shall write the entire sequence as $\mathbf{i}_0, \mathbf{i}_1, \ldots, \mathbf{i}_{L+T-1}$ where $\mathbf{i}_u = \mathbf{0}$ for $L \leqslant u < L+T$. We shall also use the convention that $\mathbf{i}_u = \mathbf{0}$ for $u < 0$.

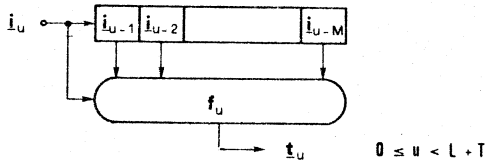

Fig. 5.1 A General (n,R,L,T,M) Trellis Encoder

With the above conventions, a general (N,R,L,T,M) trellis encoder may be represented as shown in Figure 5.1. Each square box denotes a delay of one time unit and the oval box denotes a general function from the m-ary information sequence to the set of channel input N-tuples. It should be noted that this function f_u is in general time-dependent. When $f_u = f$ for all u, $0 \leqslant u < L+T$, we call the encoder **fixed** (or "time-invariant").

A convolutional encoder is just a "linear" trellis encoder. To make this a meaningful statement, we must of course have an appropriate algebraic structure for the "information digits" and the encoded digits. To accomplish this, we suppose that the channel input alphabet is GF(q), the finite field of q elements (which requires that q be some power of a prime.) We then require $m = q^K$ so that each i_u may be taken as a K-tuple over GF(q). Notice that

$$R = \frac{\log m}{N} = \frac{K}{N} \log q \tag{5.1}$$

which is the rate in bits per channel use. It is common in coding theory to speak of K/N as the code rate but, more properly, this is the "dimensionless rate" of the code. For all our examples, we shall use $q = 2$ (the rules of GF(q) then being modulo-two arithmetic) for which case $R = K/N$.

For brevity, we hereafter write simply F rather than GF(q). A general (N,R,L,T,M) **convolutional encoder** is then representable as in Fig. 5.1 where we require $\mathbf{i}_u \in F^K$, $\mathbf{t}_u \in F^N$, and where we require the functions f_u, $0 \leqslant u < L+T$, to

be linear functions from $F^{(M+1)K}$ to F^N. We can then represent such functions f_u as

$$t_u = i_u G_0(u) + i_{u-1} G_1(u) + \ldots + i_{u-M} G_M(u) \tag{5.2}$$

where each $G_i(u)$ is a K x N matrix over F. For a fixed convolutional encoder (FCE), these matrices do not depend on u so that (5.2) becomes

$$t_u = i_u G_0 + i_{u-1} G_1 + \ldots + i_{u-M} G_M \tag{5.3}$$

where each G_j is a K x N matrix over F. In Figs. 5.2 (a) and (b), we show a general

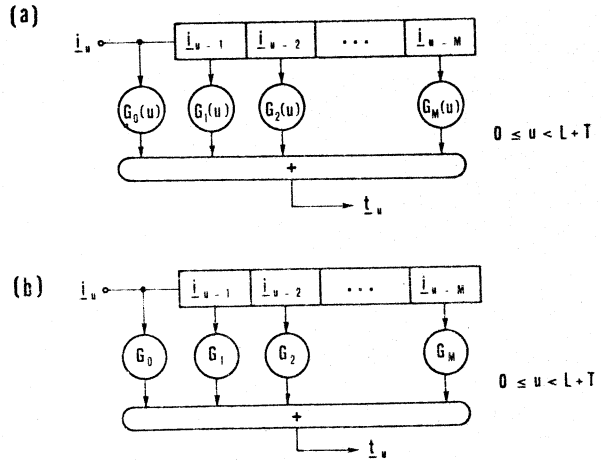

Fig. 5.2 (a) A General Convolutional Encoder and
(b) A General Fixed Convolutional Encoder (FCE).

convolutional encoder and a general FCE respectively. The name "convolutional encoder" stems from (5.3) which displays the encoded sequence as the "convolution" of the information sequence with the sequence of matrices $G_0, G_1, \ldots G_M$.

We shall find it convenient to write $i_{[u,v]}$ and $i_{[u,v)}$ for the sequences $i_u, i_{u+1}, \ldots i_v$ and $i_u, i_{u+1}, \ldots i_{v-1}$ respectively and similarly for $t_{[u,v]}$ and $t_{[u,v)}$. In this notation, the operation of a general convolutional encoder may, according to (5.2), be written as

$$\mathbf{t}_{[0,L+T)} = \mathbf{i}_{[0,L)} \begin{bmatrix} G_0(0) & G_1(1) & \cdots & G_M(M) & & & \\ & G_0(1) & \cdots & G_{M-1}(M) & G_M(M+1) & & \\ & & \ddots & \vdots & \vdots & & \\ & & & G_0(M) & G_1(M+1) & \cdots & \\ & & & & \ddots & & \\ & & & & & G_0(L-1) \cdots & G_T(L-1) \end{bmatrix} \tag{5.4}$$

where the blank portions of this matrix are assumed to be filled with zeroes. We call the matrix in (5.4) the **encoding matrix** of the (N,R,L,T,M) convolutional encoder and we denote this matrix as **G**. For the special case of a fixed convolutional encoder (FCE),

$$\mathbf{G} = \begin{bmatrix} G_0 & G_1 & \cdots & G_M & & & \\ & G_0 & \cdots & G_{M-1} & G_M & & \\ & & \ddots & & & & \\ & & & G_0 & G_1 \cdots & & \\ & & & & \ddots & & \\ & & & & & G_0 \; G_1 \cdots & G_T \end{bmatrix} \tag{5.5}$$

where there are L "rows" of matrices, each of which is a K x N matrix over $F = GF(q)$.

6. Random Coding Bounds for Convolutional Codes

We now examine the question of whether convolutional tree and trellis codes can meet the upper bounds on P_e established in Sections 3 and 4 respectively for the general classes of tree and trellis codes. We note that the symmetry of linear codes requires at once the restriction to channels which are **symmetric** in the sense that $Q(x) = 1/q$ for all x is the minimizing distribution in (2.2). Fortunately, this includes all binary input channels which are the cases of greatest practical interest. (One can handle channels for which $Q(x) = 1/q$ is not the appropriate distribution by considering convolutional codes over alphabets larger than q letters and mapping

the requisite number of letters in the larger alphabet to each q-ary letter to approximate the desired probability. In fact, q need not then be a power of a prime. We omit the awkward details of such a generalization.)

There is an artifice required with convolutional codes, as indeed with linear codes of every type, to show that some ensemble can achieve some random coding bound which arises from the fact that the all-zero information sequence is always encoded as the all-zero codeword. Thus, there is no way to pick an ensemble of convolutional codes so that, over the ensemble, the codeword assigned to the all-zero information sequence would appear to be "randomly selected" according to the given Q(x). To obviate this difficulty, one considers adding a random sequence to each codeword before transmission over the channel, i.e. the information sequence $\mathbf{i}_{[0,L+T)}$ is transmitted as the sequence $\tau_{[0,L+T)} = \mathbf{t}_{[0,L+T)}$ $\mathbf{r}_{[0,L+T)}$ where the digits in $\mathbf{r}_{[0,L+T)}$ are independently selected according to the distribution $Q(x) = 1/q$ for all x. We can write this as

$$\tau_{[0,L+T)} = \mathbf{i}_{[0,L)}\,\mathbf{G} + \mathbf{r}_{[0,L+T)} \tag{6.1}$$

which makes it clear that the same random sequence is added to each codeword for a specific code **G**. Regardless of the ensemble of convolutional codes, i.e. regardless of the probability distribution over the matrices **G**, it follows from (6.1) that any given $\mathbf{i}_{[0,L)}$ is equally likely to be encoded into any given $\tau_{[0,L+T)}$. Thus, the artifice of the "added-random-sequence" results, over the ensemble of codes, in making the encoded sequence for any given path in the tree or trellis appear to have its digits selected independently according to the distribution $Q(x) = 1/q$ for all x.

We next observe that in deriving bounds on $\overline{P}_e$ in Sections 3 and 4, we used only the independence between two "unmerged" paths in the tree or trellis. In other words, the only property our code ensembles need to ensure that these bounds on $\overline{P}_e$ hold is that the digits on any two paths diverging from some node are mutually independent over the span to the next node (if any) where the two paths join again. We call an ensemble of codes **pairwise independent** if it enjoys this property. It follows that, when we employ the added-random-sequence artifice, an ensemble of tree or trellis codes over $F = GF(q)$ will be pairwise independent if and only if the **difference** of the encoded sequences on any two paths diverging from some node is, over the ensemble of codes, a sequence in which each digit is indepently selected according to $Q(x) = 1/q$ up to the node (if any) where the paths first merge again. Now suppose that $\mathbf{i}_{[0,v)}$ and $\mathbf{i}'_{[0,v)}$ are the information sequences

describing such a pair of paths which diverge at depth u and remain unmerged at least to depth v. Then we see from (6.1) that the difference of their encoded sequences from node u to node v is the same as the encoded sequence $\mathbf{t}''_{[u,v)}$ resulting from encoding the difference sequence $\mathbf{i}''_{[0,v)} = \mathbf{i}_{[0,v)} - \mathbf{i}'_{[0,v)}$ since the random sequence cancels in the subtraction. Moreover, we note that $\mathbf{i}''_{[0,u)} = 0$ and $\mathbf{i}''_u \neq \mathbf{0}$ since the two paths diverge at node u. Further $\mathbf{i}''_{[u,v)} = [\mathbf{i}''_u, \mathbf{i}''_{u+1}, \ldots, \mathbf{i}''_{v-1}]$ can contain no run of M consecutive $\mathbf{0}$ information branches except possibly the last M of these information branches or else the paths would merge before node v. We summarize these observations as:

Lemma 6.1: An ensemble of (N,R,L,T,M) convolutional codes with an added-random-sequence is pairwise independent if and only if for every choice of u and v such that $0 \leqslant u < L$ and $u < v \leqslant L+T$ and for every choice of $\mathbf{i}_{[0,v)}$ such that $\mathbf{i}_j = \mathbf{0}$ for $j < u$, $\mathbf{i}_u \neq \mathbf{0}$, and $\mathbf{i}_{[u,v-1)}$ contains no internal run of M consecutive information K-tuples, the probability assignment on the codes in the ensemble is such that the resultant $\mathbf{t}_{[u,v)}$ is a sequence whose component digits are independently distributed and each has the distribution $Q(x) = 1/q$ all x.

As a first application of this lemma, we prove:

Theorem 6.1: The ensemble of $(N,R,L,T,M = L+T)$ fixed convolutional codes with an added-random-sequence such that each digit in each of the K x N matrices G_i, $0 \leqslant i < L+T$, is independently selected according to $Q(x) = 1/q$ is pairwise independent. Consequently, the bound (3.5) for random tree codes holds also for this ensemble when $Q(x) = 1/q$ is the minimizing distribution in (2.2).

To prove this theorem, we first use (5.4) and (5.5) to write

$$\mathbf{t}_{[u,L+T)} = [\mathbf{i}_u, \mathbf{i}_{u+1}, \ldots \mathbf{i}_{L-1}] \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_{L+T-u-1} \\ & G_0 & G_1 & \cdots & G_{L+T-u-2} \\ & & \ddots & & \vdots \\ & & & G_0 & \cdots G_T \end{bmatrix} \tag{6.2}$$

when $\mathbf{i}_j = \mathbf{0}$ for $j < u$. By Lemma 6.1, it suffices to show that $\mathbf{t}_{u+j}$ is equally likely to be any of the q^N q-ary N-tuples regardless of the values of $\mathbf{t}_u, \mathbf{t}_{u+1}, \ldots \mathbf{t}_{u+j-1}$ when $\mathbf{i}_u \neq \mathbf{0}$. (Since $M = L+T$ it is impossible to have a run of M consecutive $\mathbf{0}$ branches in $\mathbf{i}_{[u,L+T-1]}$ so this hypothesis of Lemma 6.1 will not explicitly be needed.) We see from (6.2) that $\mathbf{i}_u G_j$ will be a component of the sum for $\mathbf{t}_{u+j}$ and,

moreover, G_j has played no part in determining $\mathbf{t}_{[u,u+j)}$. Since over the ensemble of codes for any fixed choice of $G_0, \ldots G_{j-1}$ and hence of $\mathbf{t}_{[u,u+j)}$, G_j is equally likely to be any of the q^{KN} K x N q-ary matrices; thus, since $\mathbf{i}_u \neq \mathbf{0}$, $\mathbf{i}_u G_j$ is equally likely to be any q-ary N-tuple. Thus $\mathbf{t}_{u+j}$, whose value is $\mathbf{i}_u G_j$ plus a fixed vector determined by $G_0, \ldots G_{j-1}$, is also equally likely to be any q-ary N-tuple and the theorem is proved.

We see from Theorem 6.1 that **fixed** convolutional codes with $M = L+T$ are as "good" as general tree codes for transmitting information through a "symmetric" DMC, i.e. a DMC for which $Q(x) = 1/q$ is the minimizing distribution in (2.2). One might naturally expect that fixed convolutional codes for any M are as good as general (N,R,L,T,M) trellis codes but, surprisingly, this has not yet been demonstrated. We show now that "time-varying" convolutional codes are as good as general trellis codes and, in the next section, explore the reasons why fixed codes pose problems that are still unsolved.

Theorem 6.2: The ensemble of (N,R,L,T,M) convolutional codes with an added-random-sequence such that each digit in each of the matrices $G_i(u)$ for $0 \leqslant i \leqslant M$ and $0 \leqslant u < L + T$ is independently selected according to $Q(x) = 1/q$ is pairwise independent. Consequently, the bounds (4.3) and (4.5) for general random trellis codes, with parameters (N,R,L,T, M + T) and (N,R,L,T,M) respectively, hold also for this ensemble when $Q(x) = 1/q$ is the minimizing distribution in (2.2).

To prove this theorem, we first write

$$\mathbf{t}_{[u,v)} = [\mathbf{0},\ldots\mathbf{0},\mathbf{i}_u,\ldots\mathbf{i}_{v-1}] \begin{bmatrix} G_M(u) & & & \\ G_{M-1}(u) & G_M(u+1) & \ddots & \\ \vdots & \vdots & & G_M(v-1) \\ \vdots & \vdots & & \\ G_1(u) & G_2(u+1) & \ddots & \\ G_0(u) & G_1(u+1) & \ddots & G_2(v-1) \\ & G_0(u+1) & \ddots & G_1(v-1) \\ & & & G_0(v-1) \end{bmatrix} \tag{6.3}$$

where there are M 0's preceeding i_u as applying when $i_j = 0$ for $j < u$. Suppose further that $i_u \neq 0$ and that $i_{[u,v-1)}$ contains no internal span of M consecutive 0 branches. By Lemma 6.1, it suffices to show that t_{u+j} is equally likely to be any q-ary N-tuple regardless of the values of $t_u, t_{u+1}, \ldots t_{u+j-1}$. But the only matrices $G_i(k)$ affecting t_{u+j} are $G_0(u+j), G_1(u+j), \ldots G_M(u+j)$, none of which affect any of the previous transmitted branches. Moreover

$$t_{u+j} = i_{u+j} G_0(u+j) + i_{u+j-1} G_1(u+j) + \ldots + i_{u+j-M} G_M(u+j)$$

and at least one of the information branches in this expression must be non-zero, say; $i_{u+j-k} \neq 0$. Then regardless of the choice of the other matrices affecting $t_{[u,u+j]}$, the vector $i_{u+j-k} G_k(u+j)$ over all choices of $G_k(u+j)$ will take on the value of every q-ary N-tuple the same number of times. Thus, the theorem is proved.

It is interesting to compare the "sizes" of the ensembles in Theorems 6.1 and 6.2 where by "size" we mean the number of digits which must be chosen to specify a code.

To specify a code in the ensemble of fixed (N,R,L,T, $M = L + T$) convolutional codes of Theorem 6.1, we must select $(L + T)KN$ digits to specify the matrices G_i and another $(L + T)N$ digits to specify the added-random-sequence. Hence, this ensemble has size $(L + T)(K + 1)N$. Since there are $(L + T)N$ encoded digits, we can say the "per-digit size" or "complexity" of the ensemble is $K + 1$. The "complexity" in this sense is just the number of code parameters that must be selected for each digit in a codeword.

To specify a code in the ensemble of "time-varying" (N,R,L,T,M) convolutional codes of Theorem 6.2, we must select $(M+1)(L+T)KN$ digits to specify the matrices $G_i(u)$ and another $(L+T)N$ digits to specify the added-random-sequence. Hence, this ensemble has "size" $[(M+1)K+1](L+T)N$ and its "complexity" is $(M+1)K+1$.

One wonders whether good trellis codes are really more complex than good tree codes or whether we have not yet found the right bounding arguments for trellis codes. The next section will give some support to the latter view. Further support comes from noticing that our ensemble of trellis codes reaches its maximum "complexity" $(L+T)K+1$ when its memory M is its maximum value of $L + T - 1$. But it is precisely at this point that the trellis codes become tree codes and, thus Theorem 6.1 assures us that "complexity" only $K + 1$ suffices.

7. A Class of Good, Low "Complexity Convolutional Trellis Codes

In this section we demonstrate a class of (N,R,L,T,M) convolutional codes which have the pairwise independence property and have also much smaller "complexity" than the ensemble in Theorem 6.2. In Fig. 7.1, we show the encoder

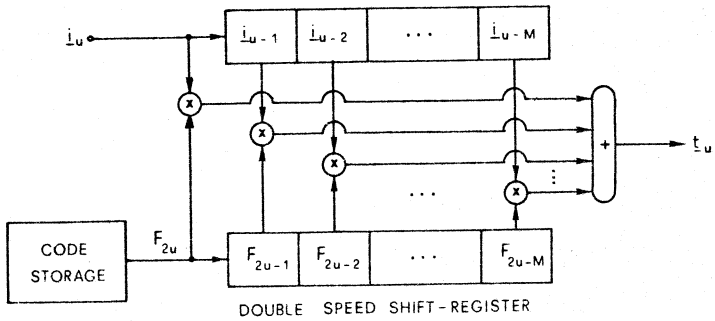

Fig. 7.1 Canonical Encoder for Low "Complexity" Trellis Codes

structure for this new class of convolutional codes. Each F_j is a K x N matrix over GF(q). The code is specified by the matrices F_j , $0 \leqslant j \leqslant 2(L+T-1)$, which are initially all in storage and fed at twice the rate of the information branches into a shift-register where the previous M of these matrices are available for multiplying the previous M information branches. In formal terms, these matrices F_j specify a time-varying convolutional code in the manner

$$G_i(u) = F_{2u-i} \tag{7.1}$$

but this formal relationship disguises the encoder structure.

For this new class of codes, we now prove:

Theorem 7.1: The ensemble of (N,R,L,T,M) convolutional codes with an added-random-sequence for which $G_i(u) = F_{2u-i}$ such that each of the digits in each of the matrices F_j , $0 \leqslant j \leqslant 2(L+T-1)$ is independently selected according to $Q(x) = 1/q$ is pairwise independent. Consequently, the bounds (4.3) and (4.5) for random trellis codes hold also for this ensemble when $Q(x) = 1/q$ is the minimizing distribution in (2.2)

To prove this theorem, we first note that, for our new class of codes, (6.3) becomes

$$\mathbf{t}_{[u,v)} = [\mathbf{0},\ldots\mathbf{0},\mathbf{i}_u,\mathbf{i}_{u+1},\ldots\mathbf{i}_{v-1}] \begin{bmatrix} F_{2u-M} & & & \\ F_{2u-M+1} & F_{2u-M+2} & \ddots & \\ \vdots & \vdots & \ddots & F_{2v-M-2} \\ F_{2u-1} & F_{2u} & \ddots & \\ F_{2u} & F_{2u+1} & \ddots & F_{2v-4} \\ & F_{2u+2} & \ddots & \\ & & \ddots & F_{2v-3} \\ & & \ddots & F_{2v-2} \end{bmatrix}$$

Again suppose that $\mathbf{i}_u \neq \mathbf{0}$ and that $\mathbf{i}_{[u,v-1)}$ contains no internal span of M consecutive $\mathbf{0}$ branches. This assumption ensures that the M +1 information branches multiplying the M + 1 non-blank matrices in each "column" of the righthand supermatrix are not all $\mathbf{0}$. Note also that the matrices F_i move diagonally upward as one moves to the right in the supermatrix. The encoded digit $\mathbf{t}_{u+j}$ is controlled by the span $\mathbf{i}_{[u+j-M,\,u+j]}$ of information branches. Let $\mathbf{i}_{u+k-M}$ be the rightmost of these branches which is not $\mathbf{0}$. This information branch multiplies a matrix F_i in the summation for $\mathbf{t}_{u+j}$ which has not affected any of the previous encoded branches since this must be the first time in its movement upward from the right that this matrix F_i has encountered a non-zero information branch. Thus, by the repeat our earlier arguments, $\mathbf{t}_{u+j}$ will be equally likely to be any q-ary N-tuple regardless of the values of $\mathbf{t}_u$, $\mathbf{t}_{u+1}$, ... $\mathbf{t}_{u+j-1}$. The pairwise independence of this ensemble of codes now follows from Lemma 6.1 and the theorem is proved.

To specify a code in this new ensemble, we must first specify the 2(L +T)KN digits in the matrices F_i. A further (L + T)N digits are required to specify the added random sequence. Hence, the ensemble has size (2K +1) (L +T)N or, equivalently, "complexity" 2K +1. This is a considerable improvement over the "complexity" (M+1)K+1 of the full ensemble of time-varying (N,R,L,T,M) codes which we showed in the previous section were

"good" in the sense of meeting the bounds on P_e of Section 4. Still, the new ensemble has a "complexity" about twice that, $K+1$, for the ensemble of (N,R,L,T, $M = L+T$) fixed convolutional codes which are "good" tree codes.

One rather mysterious aspect of this new class of "good" (N,R,L,T,M) trellis codes is that it does not include the fixed codes as a subclass. By comparison of (5.5) and (7.1), we see that the only fixed codes in the new class are those for which F_i is the same matrix for all i and these happen to be very poor codes.

The basic difficulty in trying to prove that the ensemble of fixed convolutional codes is "good" in the sense of meeting the bounds on P_e of Section 4 is that the only known way of proving such bounds for various code ensembles is via pairwise independence. Unfortunately, the ensemble of fixed convolutional codes is not a pairwise independent ensemble when $M < L+T$ so the "goodness" of fixed codes cannot be proved in the standard way. There is, however, nothing that says an ensemble must be pairwise independent to be "good" - in fact the ensemble in which one picked only the time-varying convolutional code with smallest P_e for each set of parameters N,R,L,T and M would not be pairwise independent but would surely be "good". We suspect that the ensemble of fixed (N,R,L,T,M) convolutional codes is indeed "good." The verification of this conjecture, or a proof that it is false, is an interesting open problem not without practical significance. All of the convolutionally-coded systems used in practice to date have employed fixed codes. If fixed codes are truly not as good as time-varying codes, improvement of these practical systems might be accomplished or significantly better systems might be built in the future.

8. Signal Flowcharts for Fixed Convolutional Encoders

Viterbi [3] has made clever use of signal flowchart techniques to analyze fixed convolutional encoders (FCE's). Their use is perhaps best explained through an example.

In Fig. 8.1 (a), we show an $N = 2$, $K = 1$ ($R = 1/2$) binary FCE with memory $M = 2$. We have not yet specified L or T and, in fact, the versatility of convolutional codes arises partly from the fact that these parameters can be chosen to fit the practical situation at hand. The signal flowchart for this encoder is shown in Fig. 8.1 (b) and is constructed as follows. There is a node for every "state" $[i_{u-1}, i_{u-2}]$ which is shown with the state as its label. From each state, there is a directed edge to each of the possible successor states, $[0, i_{u-1}]$ and $[1, i_{u-1}]$ for $i_u = 0$ and

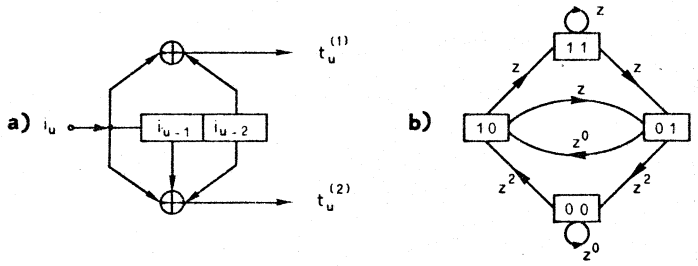

Fig. 8.1 (a) A binary FCE and (b) Its Corresponding Signal Flowchart

$i_u = 1$ respectively, which is labelled with z^W where w is the Hamming weight of the encoded branch $[t_u^{(1)}, t_u^{(2)}]$ that results from the indicated transition. For instance, when the state is $[i_{u-1}, i_{u-2}] = [1,1]$, the input $i_u = 0$ causes the encoded branch $[t_u^{(1)}, t_u^{(2)}] = [1, 0]$ and causes the next state to be $[0, 1]$. Since the Hamming weight (i.e. the number of non-zero digits) of the encoded branch is 1, the transition from state $[1, 1]$ to state $[0, 1]$ is labelled with z^1.

It should be clear from this example how one constructs the corresponding signal flowchart for any FCE. In general, since there are $(q^K)^M = q^{KM}$ states in the FCE, there will also be q^{KM} nodes in the signal flowchart so that even in the binary case ($q = 2$) the construction of the flowchart is impractical unless KM is quite small. Since the input $i_u = 0$ applied to the state $[i_{u-1}, \ldots, i_{u-M}] = [0,\ldots 0]$ always results in $t_u = 0$ for any FCE, the flowchart will always have a self-loop labelled $z^0 = 1$ at the zero state. The FCE is said to be **catastrophic** if there is any other directed loop in the flowchart whose branches are labelled $z^0 = 1$, i.e. whose "loop gain" is 1. The encoder of Fig. 8.1 (a) is non-catastrophic. The encoder of Fig. 8.2 (a) is, however, catastrophic since besides the self-loop at

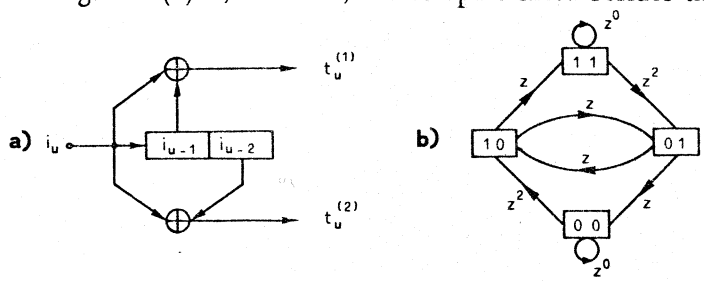

Fig. 8.2 (a) A Catastrophic Binary FCE and (b) Its Corresponding Flowchart

state $[0, 0]$ there is another directed loop with unity gain, namely the self-loop at state $[1, 1]$. (We shall later give an equivalent characterization of "catastrophic" which will make it more clear why such a pejorative term is used for this kind of FCE).

When the zero state and its self-loop are removed from the signal flowchart for an FCE, one obtains a signal flowchart with an input node (at the tails of the undeleted edges leaving the zero state) and an output node (at the points of the undeleted edges entering the zero state). There is a well-defined transmission gain, A(z), from this input node to this output node if and only if the FCE is non-catastrophic since then and only then will there be no closed loops with unity gain. The transmission gain, A(z), can be found by standard signal flowchart techniques. For example, for the flowchart in Fig. 8.1 (a), we find

$$A(z) = \frac{z^5}{1-2z} = \sum_{i=0}^{\infty} 2^i z^{5+i} \tag{8.2}$$

where the summation should be thought of as a formal power series in the indeterminate z.

We now show how to interpret the transmission gain

$$A(z) = a_0 + a_1 z + \ldots + a_i z^i + \ldots \tag{8.3}$$

obtained from the flowchart for FCE in terms of the trellis code with $L \to \infty$ specified by this FCE. For clarity, we show in Fig. 8.3 the initial portion of such a trellis for the FCE of Fig. 8.1 (a).

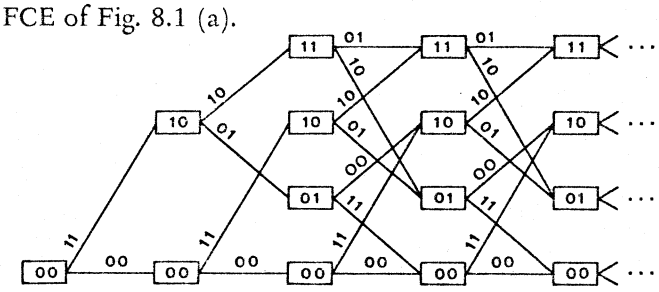

Fig. 8.3 A Portion of the $L \to \infty$ Trellis Code Defined by the FCE of Fig. 8.1 (a)

Consider any path from the input node to the output node of the flowchart with the zero state and its self-loop removed. The first branch corresponds to leaving the zero state and the last branch corresponds to the first return to the zero state. The gain z^w along this path is just the Hamming weight of this particular path through the trellis. It follows that the coefficient a_i in (8.3) is just the number of paths of Hamming weight i which depart from the all-zero lower path at the root node in the trellis and first remerge with the all-zero path at their termini. For example, from (8.2) it follows that there is only one such path of weight 5 in the trellis of Figure 8.1 as we can easily verify directly.

By the linearity of an FCE, it follows that a_i is also, for any given path through the entire trellis, the number of paths which depart from this path at the root node and first remerge with this path at their termini whose Hamming distance to the corresponding segment of the given path is exactly i. (The Hamming distance between two sequences is the number of positions in which they differ and, hence, is equal to the Hamming weight of their difference). We shall use this fact in the next section to obtain a useful upper bound on decoding error probability for a given specific FCE with MLD.

9. An Error Bound for Specific Convolutional Encoders

Consider a specific code $(\mathbf{x}_1, \mathbf{x}_2)$ containing two codewords of length N for use on a given DMC. Let $\mathbf{y} = [y_1, y_2, \ldots y_N]$ denote the received N-tuple. For a maximum likelihood decoder (MLD), the decoding region Y_1 for $\mathbf{x}_1$ is the set of all $\mathbf{y}$ such that $P(\mathbf{y}|\mathbf{x}_1) \geqslant P(\mathbf{y}|\mathbf{x}_2)$ [except that those $\mathbf{y}$ for which $P(\mathbf{y}|\mathbf{x}_1) = P(\mathbf{y}|\mathbf{x}_2)$ can be assigned arbitrarily either to Y_1 or to Y_2, the decoding region for x_2.] Now consider the decoding error probability given that $\mathbf{x}_2$ is transmitted which we shall denote $P_{e|2}$. We have

$$P_{e|2} = \sum_{\mathbf{y} \in Y_1} P(\mathbf{y}|\mathbf{x}_2). \tag{9.1}$$

Since $\sqrt{P(\mathbf{y}|\mathbf{x}_1)/P(\mathbf{y}|\mathbf{x}_2)} \geqslant 1$ for all $\mathbf{y} \in Y_1$, we can multiply each term in (9.1) by this factor to obtain

$$P_{e|2} \leqslant \sum_{\mathbf{y} \in Y_1} \sqrt{P(\mathbf{y}|\mathbf{x}_1)\, P(\mathbf{y}|\mathbf{x}_2)} \tag{9.2}$$

which we can then further overbound, by extending the summation to all $\mathbf{y}$, as

$$P_{e|2} \leqslant \sum_{\text{all } \mathbf{y}} \sqrt{P(\mathbf{y}|\mathbf{x}_1)\, P(\mathbf{y}|\mathbf{x}_2)}. \tag{9.3}$$

Since the righthand side of (9.3) is symmetric in $\mathbf{x}_1$ and $\mathbf{x}_2$, it follows that the same bound applies for $P_{e|1}$, the probability of a decoding error given that $\mathbf{x}_1$ is transmitted. But then this must also be a bound on the decoding error probability P_e for MLD regardless of the probabilities assigned to the two codewords, i.e.

$$P_e \leqslant \sum_{\text{all } y} \sqrt{P(y|x_1)\,P(y|x_2)}. \tag{9.4}$$

Using the notation of Section 2, we can rewrite (9.4) as

$$P_e \leqslant \prod_{n=1}^{N} \sum_{y_n \in B} \sqrt{P(y_n|x_{1n})\,P(y_n|x_{2n})}. \tag{9.5}$$

[The interested reader should now have no difficulty deriving the bound (2.1) by averaging the bound (9.5) over the ensemble of codes with two codewords.]

The bound (9.5) is surprisingly tight for MLD of a given code with two codewords on a given DMC. To illustrate its use, consider the binary symmetric channel (BSC) of Fig. 2.2. When $x_{1n} = x_{2n}$, the summation in (9.5) is unity. When $x_{1n} \neq x_{2n}$, the summation in (9.5) is $2\sqrt{\epsilon(1-\epsilon)}$ where ϵ is the crossover probability of the BSC. Hence, (9.5) becomes

$$P_e \leqslant [2\sqrt{\epsilon(1-\epsilon)}]^{d_H(x_1, x_2)} \tag{9.6}$$

where d_H denotes the Hamming distance between the indicated N-tuples. The simple bound (9.6) is often surprisingly close to the true P_e for MLD of the code (x_1, x_2) as the reader can verify by some detailed examples.

We note in general for any binary input channel (not just the BSC) that the bound (9.5) will take the form

$$P_e \leqslant \gamma^{d_H(x_1, x_2)} \tag{9.7}$$

where

$$\gamma = \sum_{y \in B} \sqrt{P(y|0)\,P(y|1)} \tag{9.8}$$

and $\gamma < 1$ unless the channel is "useless," i.e. unless $P(y|0) = P(y|1)$ for all y.

We now show, following Viterbi [3], how to combine the bound (9.7) with the transmission gain A(z) for a non-catastrophic FCE to obtain an upper bound on MLD of the trellis code defined by this particular FCE. More precisely, we shall upperbound the "first error probability" P_{e1} that the MLD decodes i_0

incorrectly, i.e. that the MLD chooses a path through the trellis which departs from the correct path at the root node. But an MLD will decode i_0 incorrectly if and only if there is some path departing from the correct path at the root node and first remerging with the correct path at its terminus such that the received sequence is more likely given that path than given the corresponding portion of the correct path. If this path is at Hamming distance i from the correct path then, by (9.7), the probability that it will cause $\mathbf{i}_0$ to be decoded incorrectly is bounded above by γ^i. If there are a_i such paths at Hamming distance i, then, by the union bound, the probability that they will cause $\mathbf{i}_0$ to be decoded incorrectly is bounded above by $a_i\gamma^i$. Taking into account all possible values of i with another union bound, we have finally

$$P_{el} \leqslant a_0 + a_1\gamma + a_2\gamma^2 + \ldots$$

which in light of (8.3) can be written

$$P_{el} \leqslant A(\gamma) \tag{9.9}$$

where γ is defined by (9.8). The bound (9.9) is our desired upper bound on the probability of incorrectly decoding $\mathbf{i}_0$ when a MLD is used for the trellis code defined by the binary FCE associated with A(z). Since the bound applied for $L \to \infty$, it holds for any finite L **a fortiori** provided that T = M so that every path eventually remerges with the correct path.

As a specific example, consider the BSC with $\epsilon = .01$. For this channel

$$\gamma = 2\sqrt{\epsilon(1-\epsilon)} \cong .20.$$

Combining (9.9) and (8.2), we see that the probability of decoding i_0 incorrectly, when a MLD is used with a trellis code defined by the FCE of Fig. 8.1, is overbounded by

$$P_{el} \leqslant \frac{(.20)^5}{1-2(.20)} \cong 5 \times 10^{-4}$$

which is an improvement by at least a factor of 20 over the raw "error probability" of the channel.

It has recently been observed [21] that the bound on P_{el} given by (9.9) can be improved for the BSC as follows. The bound (9.6), although quite

tight when $d_H(x_1,x_2)$ is even, can be replaced by

$$P_e \leqslant [2\sqrt{\epsilon(1-\epsilon)}]^{d_H(x_1,x_2)+1}, \quad d_H(x_1,x_2) \text{ odd}$$

since the same number of errors are required to cause a decoding error for the case of a given odd d_H as for the case when d_H is increased by 1. It follows then that, for the BSC,

$$P_{el} \leqslant a_0 + a_2\gamma^2 + a_4\gamma^4 + \ldots$$
$$+ a_1\gamma^2 + a_3\gamma^4 + \ldots$$

or

$$P_{el} \leqslant \frac{1}{2}[(1+\gamma)\,A(\gamma) + (1-\gamma)\,A(-\gamma)]. \tag{9.10}$$

For the BSC with $\epsilon = .01$ and the code with A(z) given by (8.2), we find from (9.10)

$$P_{el} \leqslant 2.3 \times 10^{-4}$$

which is an improvement on our earlier bound by a factor of more than 2. (In fact, Van de Meeberg [21] shows that (9.10) can be slightly further improved by taking $\gamma = \sqrt{2\epsilon}$).

Before closing this section, we should remark that P_{el} is also the probability of decoding i_u incorrectly given that $i_0, i_1, \ldots i_{u-1}$ have all been correctly decoded by the MLD in the case $L \to \infty$, We should also emphasize that the bound (9.9) applies for any binary input DMC and often gives a useful bound on the performance of a coding system employing a "short constraint length" convolutional code together with a MLD for such a binary input DMC.

10. Distance Measure for Fixed Convolutional Encoders

In this section, we describe some distance measures which have been proposed for fixed convolutional encoders (FCEs) and we indicate the relationship between these measures and the performance of various kinds of decoders for these codes.

The column distance of order i, denoted d_i, is defined to be the minimum

Hamming distance between two encoded paths $\mathbf{t}_{[0,i]}$ in the $L\to\infty$ trellis defined by the FCE resulting from information sequences $\mathbf{i}_{[0,i]}$ with differing values of $\mathbf{i}_0$. Equivalently, $\mathbf{d}_i$ is the minimum Hamming distance between two paths $i+1$ branches in length which diverge at the root node of the trellis. Because of the linearity of the FCE, d_i is equal to the minimum Hamming weight of an encoded path $\mathbf{t}_{[0,i]}$ resulting from an information sequence with $\mathbf{i}_0 \neq \mathbf{0}$. Letting W_H denote the Hamming weight of a sequence, we can make use of (5.4) and (5.5) to write

$$d_i = \min_{\mathbf{i}_0 \neq \mathbf{0}} W_H([\mathbf{i}_0, \mathbf{i}_1, \ldots \mathbf{i}_i] \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_i \\ & G_0 & G_1 & \cdots & G_{i-1} \\ & & \ddots & \ddots & \\ & & & & G_0 \end{bmatrix}) \tag{10.1}$$

where we recall our earlier convention that $G_j \equiv 0$ when $j > M$ and M is the memory of the FCE.

To see the reason for the terminology "column distance", we consider the semi-infinite "supermatrix" G assumed by the righthand side of (5.5) as $L\to\infty$, i.e.

$$G = \begin{bmatrix} G_0 & G_1 & \cdots & G_M & & & \\ & G_0 & \cdots & G_{M-1} & G_M & & \\ & & \ddots & & \ddots & \ddots & \\ & & & G_0 & G_1 & \cdots & G_M \\ & & & & \ddots & & \ddots \end{bmatrix} \tag{10.2}$$

When we speak of "superrows" and "supercolumns", we mean that the $K \times N$ matrices G_j are to be treated as single entries. For example, the second superrow of G is $[0, G_0\ G_1 \ldots G_M\ 0\ 0 \ldots]$. From (10.1), it follows that d_i is the minimum weight vector in the row space of the matrix formed by truncating G after $i+1$ supercolumns which includes a non-zero multiple of the first superrow in the truncated matrix.

It follows from (10.1) that d_i is a nondecreasing function of i. Moreover, every d_i is bounded above by the Hamming weight of any row in the finite matrix $[G_0\ G_1 \ldots G_M]$ so that the limit

$$d_\infty = \lim_{i\to\infty} d_i \tag{10.3}$$

always exists. It follows that

$$d_0 \leqslant d_1 \leqslant d_2 \leqslant \ldots d_i \leqslant \ldots \leqslant d_\infty . \tag{10.4}$$

The distance, d_∞, is perhaps the single most important parameter of a convolutional code for reasons we shall soon make evident. The terminology "free distance," first used by this writer, has now been generally accepted for this distance measure and one will often find the symbols d_f or d_{free} in place of d_∞.

In terms of the signal flowchart for the FCE, d_∞ is the minimum weight of an infinitely long directed path (more precisely, the minimum W for the path gain Z^W) leaving the zero state. Since d_∞ is finite, such a minimum path must return to the zero state and remain in the self-loop thereafter when the FCE is non-catastrophic. From our definition of A(z) in Section 8 for a non-catastrophic encoder, it follows that

$$d_\infty = \min \{i \mid a_i \neq 0\}. \tag{10.5}$$

Hence, as the bound (9.9) indicates, d_∞ is the main determiner of P_e for maximum likelihood decoding (MLD) of the trellis code defined by the FCE and used on a given DMC. In general, for two FCE's of the same rate and memory, the one with larger d_∞ will give the smaller P_e. The same remark applies to "almost MLD" schemes such as sequential decoding which we shall discuss later. Because of its fundamental importance, considerable effort has gone into the search for FCE's which, for a given rate R and memory M, have the largest d_∞, cf. [4] and [5] for example.

The quantity d_M is called the **feedback-decoding minimum distance**, or simply the "minimum distance," of the FCE and one often sees the alternative notations d_{FD} and d_{min}. This distance is of importance for "algebraic decoders" of the type that estimate $\mathbf{i}_0$ from the "first constrain length" $\mathbf{r}_{[0,M]}$, then "subtract" the effect of $\mathbf{i}_0$ from the received sequence and use the same algorithm to estimate $\mathbf{i}_1$ from $\mathbf{r}_{[1,M+1]}$, etc. There exists such a decoder which correctly decodes $\mathbf{i}_0$, $\mathbf{i}_1$, ..., $\mathbf{i}_u$ whenever there are t or fewer errors in each constraint length $\mathbf{r}_{[j,M+j]}$, $0 \leqslant j \leqslant u$, when and only when

$$t \leqslant \frac{1}{2}(d_M - 1). \tag{10.6}$$

This measure was first used by Wozencraft and Reiffen [6] who proved a "Gilbert lower bound" on d_M for codes of rate R = 1/N that was later generalized by this writer to all rates [7].

The $(M+1)$-tuple $\mathbf{d} = [d_0, d_1, \ldots, d_M]$ has been recently introduced [5] and called the **distance profile** of the FCE. If $\mathbf{d}$ and $\mathbf{d}'$ are distance profiles for two FCE's of the same rate and memory, one says $\mathbf{d} > \mathbf{d}'$ if $d_j > d_j'$ for the smallest index j, $0 \leqslant j \leqslant M$, where $d_j \neq d_j'$. The code with the larger distance profile will have a minimal separation between paths diverging at the root node which grows more rapidly, at least initially, with depth into the trellis. For this reason, the code with the larger $\mathbf{d}$ will generally require less computation when sequential decoding is used than will the other code.

The **row distance of order** i, denoted r_i, is defined to be the minimum Hamming distance between two different paths in the $(N, R, L+i+1, T=M, M)$ trellis defined by the FCE. By the linearity of the FCE, it follows that

$$r_i = \min_{\mathbf{i}_{[0,i]} \neq 0} W_H\left([\mathbf{i}_0, \mathbf{i}_1, \ldots, \mathbf{i}_i] \begin{bmatrix} G_0 & G_1 & \ldots & G_M & & \\ & G_0 & G_1 & \ldots & G_M & \\ & & \ddots & \ddots & & \ddots \\ & & & G_0 & G_1 & \ldots & G_M \end{bmatrix}\right) \tag{10.7}$$

which is equivalent to saving that r_i is the minimum weight of non-trivial linear combinations of the rows in the matrix formed by truncating the super-matrix G after its first $i+1$ superrows. In terms of the signal flowchart of the FCE, r_i is the minimum weight of a path of length $M+i+1$ branches which diverges at some point from the zero state and terminates on the zero state.

It follows from (10.7) that r_i is non-increasing with i. Since every r_i is bounded below by 0, it follows that the limit

$$r_\infty = \lim_{i \to \infty} r_i$$

always exists. Thus, we have

$$r_0 \geqslant r_1 \geqslant r_2 \geqslant \ldots \geqslant r_i \geqslant \ldots \geqslant r_\infty \tag{10.8}$$

which should be contrasted with (10.4). Moreover, we see from (10.1) and (10.7) that

$$d_i \leqslant r_j \quad \text{all i and j.}$$

From (10.4) and (10.8), we then have the inequalities

$$d_0 \leqslant d_1 \leqslant \ldots \leqslant d_i \leqslant \ldots \leqslant d_\infty \leqslant r_\infty \leqslant \ldots \leqslant r_i \leqslant \ldots \leqslant r_1 \leqslant r_0 \qquad (10.9)$$

which lie at the heart of distance measures for FCE's. By the relationship we noted between the distance measures d_i and r_j and paths in the signal flowchart for the FCE, it follows that

$$d_\infty = r_\infty \qquad (10.10)$$

for a non-catastrophic FCE.

From its definition, we see that r_i specifies the error-correcting property of the $L = i + 1$, $T = M$ trellis generated by the FCE in that there exists a decoder that can correct all patterns of t or fewer errors in the full encoded sequence if and only if

$$t \leqslant \frac{1}{2}(r_i - 1). \qquad (10.11)$$

Nonetheless, the row distances r_i are not of much direct interest in convolutional coding for the reasons (1) that in practice the trellis length L is usually great enough that r_∞ is the appropriate distance, and (2) that in practice one always chooses a non-catastrophic FCE so that $r_\infty = d_\infty$. The main utility of the row distance is in assisting the calculation or bounding of the column distances via (10.9). For example, from the flowchart for the FCE in Fig. 8.1, we readily find in order that $d_0=2, d_1=3, d_2=3, d_3=4, d_4=4, d_5=5, r_0=5$. From (10.9) it then follows immediately that $d_i=5$ for all $i \geqslant 5$ (and $r_i=5$ for all $i \geqslant 0$). In particular, the free distance of this code is $d_\infty = 5$. It should be noted that (10.10) may not hold when the FCE is catastrophic. For example, from the flowchart for the FCE in Fig. 8.2, we find that $d_0 = 2$ and that $d_i = 3$ for all $i \geqslant 1$ so that $d_\infty = 3$. On the other hand, we find that $r_i = 4$ for all $i \geqslant 0$ so that $r_\infty = 4 > d_\infty = 3$.

For an intersting discussion of the distance measures considered here as well as their values for many specific FCE's, the reader is referred to the paper by Costello [8]. The paper by Johannesson [5] is a good source for excellent FCE's of rate $R = 1/2$ which is the rate most often used in practice.

11. Catastrophic Fixed Convolutional Encoders

In this section, we probe more deeply into the meaning of a catastrophic FCE as defined previously in Section 8 in terms of the signal flowchart of the FCE. We first note that our definition there is equivalent to saying that an FCE is catastrophic if and only if there is an information sequence $\mathbf{i}_{[0,\infty)}$ with $W_H(\mathbf{i}_{[0,\infty)}) = \infty$ that produces an encoded sequence $\mathbf{t}_{[0,\infty)}$ for which $W_H(\mathbf{t}_{[0,\infty)}) < \infty$. To see this equivalence, we note that if the signal flowgraph has a closed loop of weight 0 besides the self-loop at the zero state, then the information sequence $\mathbf{i}_{[0,\infty)}$ which drives the encoder to a state on the former loop and around the loop forever after is an infinite weight sequence whcih produces a finite weight encoded sequence. Conversely, if there is no zero weight loop besides the self-loop at the zero state, then an information sequence of infinite weight must drive the encoder through infinitely many loops besides the self-loop at the zero state and hence must produce an encoded sequence of infinite weight also.

We now give further characterizations of catastrophic FCE's arising from the fact that an FCE is a K-input, N-output linear sequential circuit (LSC). In fact, an FCE is an LSC with input memory M in the sense that its output depends only on the present input and the M preceeding inputs. An LSC with finite input memory is called feedforward (FF) and is distinguished by the fact that the entries in its transfer function matrix (written in the delay operator D) are all polynomials. For example, the K = 1, N = 2, M = 2 FCE in Fig. 8.1 has the K x N transfer function matrix

$$\mathbf{T}(D) = [1 + D^2 \;\; 1 + D + D^2] \tag{11.1}$$

as can easily be read from its defining circuit.

A second LSC with transfer function $\mathbf{T}^*(D)$ is said to be a delay-Δ inverse of the LSC with transfer function T(D) whenever (when both are started in the zero state) the output sequence of the original LSC produces, when used as the input sequence to the inverse, an output sequence equal to the input sequence of the original except for a delay of Δ time units, that is when

$$\mathbf{T}(D)\,\mathbf{T}^*(D) = D^{\Delta}. \tag{11.2}$$

For instance, the simple LSC with transfer function matrix

$$\mathbf{T}^*(D) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{11.3}$$

is a delay $\Delta = 1$ inverse of the FCE of Fig. 8.1 whose transfer function is given in (11.1). Similarly the LSC with transfer function

$$T^*(D) = \begin{bmatrix} 1 + D \\ D \end{bmatrix} \tag{11.4}$$

is a delay $\Delta = 0$ inverse, i.e. an **instantaneous inverse**, of the same FCE.

An LSC is called **feedforward-invertible** if it has a delay-Δ inverse, for some Δ, which is a feedforward LSC. As either of the above inverses for the FCE in Fig. 8.1 is feedforward-invertible. Forney [9] has shown that an LSC is feedforward-invertible if and only if there is no infinite weight input sequence which produces a finite weight output sequence. From our second characterization above of a catastrophic FCE, a third characterization immediately follows; namely that an FCE is **catastrophic if and only if it is not feedforward-invertible.**

Massey and Sain have shown that an FCE, or more generally any FF LSC, is feedforward-invertible if and only if the greatest common divisor of the K x K minors of its transfer function matrix is a monomial, i.e. of the form D^i for some i [10]. Hence, the FCE of Fig. 8.1 with $K = 1$ and $N = 2$ could be recognized as non-catastrophic from the fact that the polynomials in its transfer function matrix of (11.1) are relatively prime, that is, their greatest common divisor is 1. On the other hand, the transfer function matrix of the FCE of Fig. 8.2 is

$$\mathbf{T}(D) = [1 + D \;\; 1 + D^2] \tag{11.5}$$

and, since $(1 + D)^2 = 1 + D^2$, from the fact that the greatest common divisor of its entries is $1 + D$ one can conclude that this FCE is catastrophic. (Olson has extended this test for feedforward invertibility to arbitrary LSC's [11] and Forney [9] has given an elegant algebraic formulation of Olson's test.)

If one divides each entry of the transfer function matrix of (11.5) for the catastrophic FCE of Fig. 8.2 by the greatest common divisor $1 + D$ of its entries, one obtains the transfer function matrix for the simpler $M = 1$ FCE of Fig. 11.1. One readily finds that $d_0 = 2$ and $d_i = 3$ for $i \geqslant 1$ just as was the case for the catastrophic FCE from which the FCE of Fig. 11.1 was obtained. As the reader may

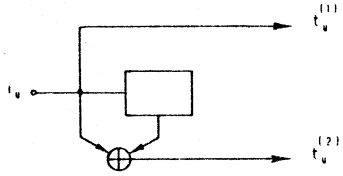

Fig. 11.1 The Non-Catastrophic FCE "Equivalent" to the Catastrophic FCE of Fig. 8.2.

suspect, this preservation of column distances was no accident – for any catastrophic FCE one can find an "equivalent" non-catastrophic FCE with the same column distances and perhaps less memory. This, in itself, is perhaps reason enough to eschew catastrophic FCE's.

While we have given three equivalent characterizations of catastrophic FCE's, the reader may not consider any of these ways of viewing the property as reason enough for the strong term "catastrophic." The fourth and last of our characterizations (which historically was the first characterization of the "catastrophic" property [10]) should make it clear why such a damning adjective is appropriate.

Consider the $L \to \infty$ trellis code generated by an FCE. A decoder is just a device for forming an estimate $\hat{\mathbf{i}}_{[0,\infty)}$ of the information sequence $\mathbf{i}_{[0,\infty)}$. The difference sequence

$$\boldsymbol{\delta}_{[0,\infty)} = \hat{\mathbf{i}}_{[0,\infty)} - \mathbf{i}_{[0,\infty)} \tag{11.6}$$

is just the sequence of decoded information errors. Now any decoder can be thought of as also making an estimate $\hat{\mathbf{t}}_{[0,\infty)}$ of the encoded sequence $\mathbf{t}_{[0,\infty)}$ transmitted over the channel simply by considering $\hat{\mathbf{t}}_{[0,\infty)}$ to be the result of encoding $\hat{\mathbf{i}}_{[0,\infty)}$. The difference sequence

$$\boldsymbol{\epsilon}_{[0,\infty)} = \hat{\mathbf{t}}_{[0,\infty)} - \mathbf{t}_{[0,\infty)} \tag{11.7}$$

in the sequence of decoded transmission errors. In fact, we can show the general situation as in Fig. 11.2 where it is clear that which encoder inverse is used is of no

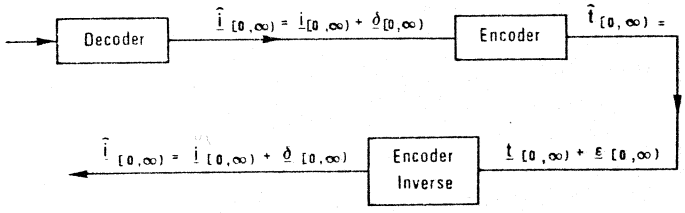

Fig. 11.2 A Canonic "Decomposition" of the Decoder for a FCE.

significance. By the linearity of the FCE and its inverse, it follows that the sequence $\boldsymbol{\delta}_{[0,\infty)}$ is the response of the encoder inverse to the sequence $\boldsymbol{\epsilon}_{[0,\infty)}$ – which

fact we shall shortly use in a critical way. One can think of the decoder-encoder tandem in Fig. 11.2 as forming a "channel sequence estimator" whose output is inverted to yield $\hat{\mathbf{i}}_{[0,\infty)}$. In fact, as we shall see later, Viterbi decoders and sequential decoders truly perform this channel sequence estimation prior to estimating the information digits.

We think the reader would agree that it would be a "catastrophic" situation if the channel sequence estimator should make only a finite number of errors, i.e. $W_H(\epsilon_{[0,\infty)}) < \infty$, but these resulted in an avalanche of infinitely many information decoding errors, i.e. $W_H(\delta_{[0,\infty)}) = \infty$. But we now claim that an FCE is catastrophic if and only if every "realistic" channel-decoder pair are such that some channel behavior will cause $W_H(\epsilon_{[0,\infty)}) < \infty$ but result in $W_H(\delta_{[0,\infty)}) = \infty$. Conversely if an FCE is non-catastrophic, no channel-decoder pair can ever result in $W_H(\epsilon_{[0,\infty)}) < \infty$ but $W_H(\delta_{[0,\infty)}) = \infty$. By a "realistic" channel-decoder pair we mean a pair such that, regardless of what $\mathbf{i}_{[0,\infty)}$ is encoded, the channel can behave so as to cause the decoder to decide $\hat{\mathbf{i}}_{[0,\infty)} = \mathbf{0}$. This rules out, for instance, the "noisless" BSC with 0 crossover probability and other "perfect" channels, and also rules out decoders that never estimate $\mathbf{0}$ but rules out no combination of a "real" channel and a "reasonable" decoder.

We prove the converse first because it is so simple. Suppose the FCE is non-catastrophic. It then has an FF inverse which we can assume is the inverse in Fig. 11.2 since it is immaterial which inverse is used. Suppose $W_H(\epsilon_{[0,\infty)}) < \infty$ and recall that $\delta_{[0,\infty)}$ is the response of the FF inverse to $\epsilon_{[0,\infty)}$. Since the FF inverse has finite input memory, after a finite time it will be see only zero digits in the sequence $\epsilon_{[0,\infty)}$ and its output must be zero thereafter. We conclude that $W_H(\delta_{[0,\infty)}) < \infty$ which proves the converse.

Now suppose that the FCE is catastrophic. There is then an input sequence $\mathbf{i}_{[0,\infty)}$ with $W_H(\mathbf{i}_{[0,\infty)}) = \infty$ that causes a transmitted sequence $\mathbf{t}_{[0,\infty)}$ with $W_H(\mathbf{t}_{[0,\infty)}) < \infty$. Next, suppose the channel behaves so as to cause $\hat{\mathbf{i}}_{[0,\infty)} = \mathbf{0}$ which implies in turn that $\hat{\mathbf{t}}_{[0,\infty)} = \mathbf{0}$. We then have $W_H(\epsilon) = W_H(\mathbf{0} - \mathbf{t}_{[0,\infty)}) = W_H(\mathbf{t}_{[0,\infty)}) < \infty$ but we also have $W_H(\delta) = W_H(\mathbf{0} - \mathbf{i}_{[0,\infty)}) = W_H(\mathbf{i}_{[0,\infty)}) = \infty$. Thus, a finite number of errors in estimating the channel sequence have been converted to an infinite number of errors in decoding the information digits and our claim is proved in full.

12. MLD (Viterbi Decoding) of Trellis Codes

In this section, we study how one might perform, in an efficient way, maximum likelihood decoding (MLD) for an arbitrary (N,R,L,T,M) trellis code used on an arbitrary discrete memoryless channel (DMC). The procedures that will be developed apply, of course, to convolutional codes which are the linear special case of trellis codes.

Suppose that $\mathbf{y}_{[0,L+T)}$ is the sequence received over the channel where (in what should now be familiar notation)

$$\mathbf{y}_{[0,L+T)} = [\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{L+T-1}]$$

and where each $\mathbf{y}_u$ is an N-tuple of channel output letters. Similarly, we write $\mathbf{x}_{[0,L+T)}$ for the encoded sequence, i.e. the sequence of channel input letters on the path through the trellis, resulting from the information sequence $\mathbf{i}_{[0,L+T)}$. A maximum likelihood decoder chooses as its estimate $\hat{\mathbf{i}}_{[0,L+T)}$ (one of) the sequence(s) which maximizes

$$P(\mathbf{y}_{[0,L+T)} | \mathbf{x}_{[0,L+T)}) = \prod_{u=0}^{L+T-1} P(\mathbf{y}_u | \mathbf{x}_u)$$

or, equivalently, which maximizes the statistic

$$\log P(\mathbf{y}_{[0,L+T)} | \mathbf{x}_{[0,L+T)}) = \sum_{u=0}^{L+T-1} \log P(\mathbf{y}_u | \mathbf{x}_u) \,. \tag{12.1}$$

Since $\mathbf{y}$ will be fixed throughout our discussion, we shall write simply

$$L_0(\mathbf{x}_{[u,v]}) = \log P(\mathbf{y}_{[u,v]} | \mathbf{x}_{[u,v]}) \tag{12.2}$$

and we note that

$$L_0(\mathbf{x}_{[u,v]}) = \sum_{i=u}^{v} L_0(\mathbf{x}_i) \,. \tag{12.3}$$

In fact, we shall not insist that L_0 be the logarithmic function as in (12.2), but simply that it be a statistic whose maximization yields a MLD and which is additive

according to (12.3). For instance, for the BSC, we can take

$$L_0(\mathbf{x}_i) = -d_H(\mathbf{x}_i, \mathbf{y}_i)$$

i.e. the negative of the number of channel crossovers or "errors" had $\mathbf{x}_i$ actually been transmitted.

The key to MLD for (N,R,L,T,M) trellis codes is the following:

Principle of Non-Optimality: If the paths $\mathbf{i}'_{[0,u]}$ and $\mathbf{i}''_{[0,u]}$ terminate at the same node of the trellis and

$$L_0(\mathbf{x}'_{[0,u]}) > L_0(\mathbf{x}''_{[0,u]}), \tag{12.4}$$

then $\mathbf{i}''_{[0,u]}$ cannot be the first u + 1 branches of (one of) the path(s) $\mathbf{i}_{[0,L+T)}$ which maximizes $L_0(\mathbf{x}_{[0,L+T)})$.

The proof of this property is very simple. Suppose on the contrary that $\mathbf{i}''_{[0,L+T)} = \mathbf{i}''_{[0,u]} * \mathbf{i}''_{(u,L+T)}$ [where here and after * denotes concatenation of sequences] maximizes the statistic L_0. Then consider the path $\mathbf{i}'_{[0,L+T)} = \mathbf{i}'_{[0,u]} * \mathbf{i}''_{(u,L+T)}$. The encoded sequence $\mathbf{x}'_{[0,L+T)} = \mathbf{x}'_{[0,u]} * \mathbf{x}_{(u,L+T)}$ must have $\mathbf{x}'_{(u,L+T)} = \mathbf{x}''_{(u,L+T)}$ since $\mathbf{i}'_{[0,u]}$ terminates on the same node as $\mathbf{i}''_{[0,u]}$ and we have used the same subsequent information sequence $\mathbf{i}''_{(u,L+T)}$ in both cases. Consequently,

$$\begin{aligned} L_0(\mathbf{x}'_{[0,L+T)}) &= L_0(\mathbf{x}'_{[0,u]} * \mathbf{x}''_{(u,L+T)}) \\ &= L_0(\mathbf{x}'_{[0,u]}) + L_0(\mathbf{x}''_{(u,L+T)}) \\ &> L_0(\mathbf{x}''_{[0,u]}) + L_0(\mathbf{x}''_{(u,L+T)}) \end{aligned}$$

by (12.4). Thus,

$$L_0(\mathbf{x}'_{[0,L+T)}) > L_0(\mathbf{x}''_{[0,L+T]})$$

which contradicts the assumption that $\mathbf{x}''_{[0,L+T)}$ maximized the statistic L_0.

We now show, in an example, how the principle of Non-Optimality specifies an efficient MLD procedure for trellis codes. In Fig. 12.1, we show the (n = 2, R = 1/2, L = 4, T = 2, M = 2) trellis code for the binary FCE of Fig. 8.1 and the received sequence $\mathbf{y}_{[0,6)}$ = [1,0,1,1,0,1,0,1,1,0,0,0] which has been received over a BSC with crossover probability $\epsilon < 1/2$. For our decoding statistic L_0, we use the negative of the number of errors between this received sequence and the

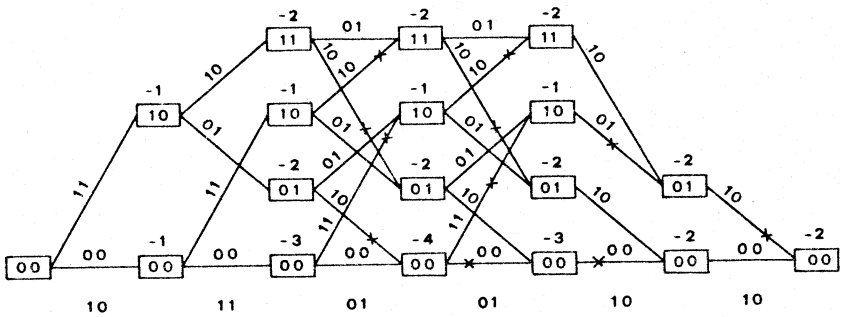

Fig. 12.1 An Example of MLD for the BSC

encoded sequence. As will soon be apparent, the number written above each node or "encoder state" is the statistic L_0 for the best path to that state.

For the nodes at depth 1 and 2 in the trellis, there is only one path to each node so there is no doubt about the best path to these nodes. At depth 3, the interesting situations begin. Of the two paths entering state [1, 1] we see that the upper has $L_0 =, -2$ and the lower has $L_0 = -3$. By the principle of non-optimality, we can discard the lower path in our search for the best path through the trellis. We show this figuratively in Fig. 12.1 by a "cross" on the last branch of the lower path to indicate we have "cut" the path at this point to leave the upper path as the only remaining path into state [1, 1]. We repeat this same procedure for the other 3 states at depth 3, and above each state we show the value of L_0 for the one remaining path to that state. Since there is now exactly one path into each state at depth 3, there are exactly two paths now into each state at depth 4. We process each state at depth 4 by "cutting" the worse path into that state. We then have exactly one path into each state at depth 4 and hence exactly 2 paths again into each state at depth 5. We then process both states at depth 5 and thus create exactly two paths into the only state, [0, 0] at depth 6. We then process this state by "cutting" the worse path into it. We now find we are left with exactly one path through the entire trellis, namely the path (which is most easily read in reverse order from Fig. 12.1) $\mathbf{x}_{[0,6)} = [0,0,1,1,0,1,1,1,0,0,0,0]$ which is the encoding of $\mathbf{i}_{[0,6)} = [0,1,0,0,0,0]$. 0,1,0,0,0,0 . Since we have been true to the principal of non-optimality and have discarded only paths which could not be optimal, we conclude that this remaining path is the optimal one. The MLD decision is thus $\hat{\mathbf{i}}_{[0,L)} = \hat{\mathbf{i}}_{[0,4)} = [0,1,0,0]$.

It should be clear that we have now found a general method to do MLD for any (N,R,L,T,M) trellis code on any DMC. Up to depth M, there will be a unique

path to each state and we can directly assign the statistic L_0. At depth M +1, there will be exactly $m = 2^{NR}$ paths into each state. We discard or "cut" all but the best one of these and assign the statistic of this best path to that state. We thus again create exactly m paths into each state at depth M + 2 and we process these states in the same manner. When we finally reach depth L + T of the tree, our final processing creates exactly one path into each of these m^{M-T}, terminal states (and in practice one would always use M = T with MLD.) That one of these m^{M-T} paths through the entire trellis which has the greatest statistic L must be the optimal path, i.e. the path to be chosen by a MLD.

This method of doing MLD for the trellis codes was introduced by Viterbi [12] and is now usually called "Viterbi Decoding." Viterbi at that time used this decoding procedure as a theoretical tool to prove the bound (4.4) for time-varying convolutional codes (which he also extended appropriately for rates R, $R_0 \leqslant R < C$,) but he was unaware at the time that his procedure was actually maximum-likelihood. That observation was first made by Omura [13] who also observed that Viterbi's algorithm was in fact the "dynamic programming" solution to the problem of optimal decoding. [What we have called the "Principle of Non-Optimality" is what Bellman calls the "Principle of Optimality" in dynamic programming but the former name strikes us as more illuminating in the present context.] The reader may wonder that these observations were made so late but he should know that Forney's use of "trellis diagrams" for convolutional codes, which makes these facts so transparent now, came considerably after Omura's work.

13. Implementing Viterbi Decoders

In the previous section, we gave the conceptual formulation of Viterbi decoding, i.e. MLD, for trellis codes. In this section, we consider how one might actually construct Viterbi decoders, either in hardware or computer software, for trellis codes generated by fixed convolutional encoders.

Because of their practical importance and conceptual simplicity, we shall consider only (N, R = 1/N, L, T = M, M) trellis codes for binary fixed convolutional encoders (FCE's). Although we shall be thus restricting ourselves to binary input channels, it is important to note that we have no corresponding restriction on the size of the channel output alphabet. "Binary output quantization" often severely degrades a channel in terms of its resultant R_0 or its resultant capacity. As will be apparent, Viterbi decoders can quite easily handle "high-order output quantization"

of the physical channel.

Since $R = 1/N$, there are only $2^{RN} = 2$ branches entering each state in the trellis, at depth $M+1$ and greater, and there are a total of 2^M distinct states. A state at depth u in the trellis may be denoted $[i_{u-1}, i_{u-2}, \ldots i_{u-M}]$ where each i_j is a binary information digit.

Consider now the amount of "processing" done at each depth u, $M < u \leqslant L$ where all 2^M states are present in the trellis and each state has two branches leading into it. At such a given time u, the previous processing in the Viterbi algorithm will ensure that there are only two paths from the root node to each state at time u. For each such state, it is necessary to compare these two paths, discard the worse, compute the statistic L_0 on each of the two branches leading out of the state, and add this branch statistic to the statistic for the better of the two paths before sending these values on to the two successor states at depth $u+1$. To accomplish this task in hardware, it is natural to consider using a separate "microcomputer" for each of the 2^M states.

Let $M(b_1, b_2, \ldots b_M)$ denote the "microcomputer" corresponding to state $[b_1, b_2, \ldots b_M]$. We can think of each microcomputer functioning at the u-th step of the computation, $u = 0, 1, 2, \ldots L+M$, according to the diagram shown in Fig. 13.2. At step u, the microcomputer $M(b_1, b_2, \ldots b_M)$ receives the output of two

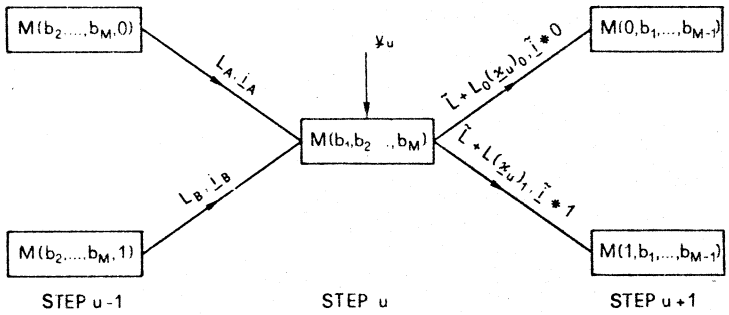

Fig. 13.2 Functioning of Microcomputers in a Hardware Viterbi Decoder

designated microcomputers from step u - 1 consisting of the statistic (L_A or L_B) and the information sequence (i_A or i_B) for the two surviving paths into state $[b_1, b_2, \ldots b_M]$. The microcomputer $M(b_1, b_2, \ldots b_M)$ first compares these statistics and sets

$$\tilde{L} = L_A \text{ and } \tilde{i} = i_A \text{ if } L_A \geqslant L_B$$
$$\tilde{L} = L_B \text{ and } \tilde{i} = i_B \text{ if } L_A < L_B.$$

Since the encoded branch $\mathbf{x}_u$ is determined uniquely by the state and the information digit i_u, the labels on these branches, say $(\mathbf{x}_u)_0$ and $(\mathbf{x}_u)_1$ for $i_u = 0$ and $i_u = 1$ respectively, are always the same for a given microcomputer $M(b_1, b_2, \ldots b_M)$. Hence, given the received branch $\mathbf{y}_u$, this microcomputer can immediately determine (say by look-up in a stored table) the appropriate branch metrics, say $L_0(\mathbf{x}_u)_0$ and $L_0(\mathbf{x}_u)_1$ respectively. The microcomputer then adds these branch statistics to the statistic $\tilde{L}$, appends the appropriate information digit to $\tilde{i}$, and sends these results along to two designated microcomputers for their use in step $u+1$ of the computation. Thus each "microcomputer" is a rather simple digital device.

At the steps u, $u \leqslant M$ and $u > L$, we see that in principle not all microcomputers should be in operation since not all 2^M states are then present in the trellis or not all have 2 input branches. A simple trick avoids the complication of "activating" and "deactivating" certain microcomputers; at step 0 one initializes all microcomputers, except M(0,0,...0) which is initialized with $\tilde{L} = 0$, with sufficiently negagive L so that the path from these states can never be the final chosen path and, at $u = L + M$ when the computation is finished one ignores the output of all microcomputers except M(0,0,...0) whose output $\tilde{i}$ is taken as the decoded estimate. In this manner, all microcomputers may be left to function at all times which simplifies the controlling logic.

It should be pointed out that there are only 2^N different types of microcomputers, regardless of the value of M, for FCE's in which $(\mathbf{x}_u)_0$ is the complement of $(\mathbf{x}_u)_1$ [as is the case with all "good" $R = 1/N$ binary encoders] since the number of different microcomputers is then the number of possible encoded branches $(\mathbf{x}_u)_0$ of N binary digits. For instance, for $R = 1/2$ which is the most important case in practice, there are only $2^2 = 4$ different types of microcomputers. Hence, a hardware Viterbi decoder for an $M = 6$, $R = 1/2$ FCE would have $2^6 = 64$ microcomputers but only 4 distinct types of microcomputers.

When a Viterbi decoder is implemented in software on a general-purpose digital computer, the usual method is to perform serially, at step u, the operations of each of the 2^M microcomputers of a hardware realization.

Whether implemented in hardware or software, it is clear that the "complexity" of a Viterbi decoder grows exponentially with M, the encoder memory. $M = 6$ or 7 appears to be about the limit of practicality. Rate 1/2 FCE's with memory $M \cong 6$ are, however, surprisingly good for their short constraint lengths $N_t = (M+1)N \cong 14$. In fact, at the present writing, if one is content with

an error probabability of 10^{-4} or greater, Viterbi decoders operate at as small or smaller energy per information bit to noise power spectral density ratios on the "deep-space channel" as any other practical coding system and are rather widely used in space applications. The prospect is for even wider use of Viterbi decoders in future digital communication systems [3].

14. Decoding Tree Codes--The Fano Metric

We now take up the question of how to decode tree codes in a reasonable way. We can of course consider FCE's to generate a tree code as well as a trellis code so our results will apply, in particular, to convolutional codes. (Generally speaking, we choose to view FCE's as generating tree codes rather than trellis codes when M is so large that a Viterbi-type trellis decoder is impractical.)

When L is quite large, the number of paths 2^{NRL} in a tree code is so great that one cannot conceive of comparing each path with the received sequence to perform maximum-likelihood decoding (MLD). For instance, with $R = 1/N$ and $L = 100$ (which is "small" as practical tree codes go) there are $2^{100} \cong 10^{30}$ paths through the tree. A feeling for the size of this number can be obtained by reflecting that 10^{30} is about one million times greater than Avogadro's number! But since large numbers of paths stem from each of the early nodes in the tree, one suspects that it might be possible to discard all the paths stemming from such a node when the path to it is sufficiently bad without severely degrading performance from that of MLD. This, in fact, is the whole point of using tree codes. But to discard paths as "bad," we need some absolute measure of their quality which will take into account the fact that the paths we discard may of different lengths. To obtain such a measure of "quality metric" we are led naturally to consider the decoding problem for codes whose codewords (unlike those of block codes) have different lengths.

Let $(x_1, x_2, \ldots x_S)$ be a code with S codewords,

$$\mathbf{x}_s = [x_{s1}, x_{s2}, \ldots x_{sN_s}] \quad s = 1,2,\ldots S$$

whose lengths $N_1, N_2, \ldots N_s$ are in general different. Consider transmitting the message s by sending $\mathbf{x}_s$ over a DMC. The decoder must make its estimate $\hat{s}$ of which message was sent. Let

$$N = \max(N_1, N_2, \ldots N_S)$$

and we require the decoder to estimate s from the received sequence $\mathbf{y} = [y_1, y_2, \ldots y_N]$.

To avoid any information about s being sent over the DMC covertly after the codeword $\mathbf{x}_s$ has been transmitted and when $N_s < N$, we suppose that a random tail of $N - N_s$ digits obtained by independent selection from the channel input alphabet according to a probability distribution $Q(x)$ are appended to $\mathbf{x}_s$ for transmission over the DMC and that the decoder knows only the distribution $Q(x)$ used in this selection. We do not wish to assume the messages are equiprobable so we let P_s denote the probability that message s is sent. We then seek to find a decoding rule which minimizes the probability of a decoding error.

Let $P(s, \mathbf{y})$ denote the joint probability of sending message s and receiving $\mathbf{y}$. Since our channel is memoryless and the random digits following $\mathbf{x}_s$ are independently chosen, we have

$$P(s,\mathbf{y}) = P_s \prod_{n=1}^{N_s} P(y_i \mid x_{si}) \prod_{t=N_s+1}^{N} P_Q(y_t) \tag{14.1}$$

where $P_Q(y)$ is the probability of receiving y given that a random digit is sent over the channel, i.e.

$$P_Q(y) = \sum_{x \in A} P(y \mid x)\, Q(x) \tag{14.2}$$

where A is the channel input alphabet. The probability of a decoding error is minimized by choosing $\hat{s}$ as the value of s which maximizes $P(s, \mathbf{y})$, or, equivalently, which maximizes

$$\frac{P(s,\mathbf{y})}{\prod_{i=1}^{N} P_Q(y_i)} = P_s \prod_{n=1}^{N_s} \frac{P(y_n \mid x_{sn})}{P_Q(y_n)} \tag{14.3}$$

as we have merely divided by a positive constant which is independent of s. Again equivalently, we may take logarithms in (14.3) to assert that the probability of a decoding error will be minimized by choosing $\hat{s}$ as that value of s which maximizes the statistic

$$L_f(s,\mathbf{y}) = \sum_{n=1}^{N_s} \left[\log \frac{P(y_n \mid x_{sn})}{P_Q(y_n)} - \frac{1}{N_s} \log \frac{1}{P_s}\right] \tag{14.4}$$

We note the somewhat surprising but comforting fact that this "decoding metric" for message s depends only on that part of **y** of the same length as s.

Now consider how we can use the above metric to perform "almost MLD" of tree codes without exploring the entire code tree. As a specific instance, consider an $R = 1/N$ tree code with $N = 3$ for a binary input channel. Suppose that we have partially explored the tree as shown in Fig. 14.1. Which of the 4 terminal nodes should we extend to continue an efficient search for the same encoded path that a MLD would find at least "most of the time"? Recall that a MLD is equivalent to a minimum error probability decoder for equiprobable codewords. But assuming that the 2^{NRL} codewords in a tree code are equally likely is the same as assuming that, at any dividing node, the encoder follows any of the 2^{NR} branches with probability 2^{-NR}. Hence, we should assume that messages 1, 2, 3, and 4 in Fig. 14.1 have probabilities $P_1 = 2^{-1}$, $P_2 = P_3 = 2^{-3}$ and $P_4 = 2^{-2}$. We should then use these probabilities in (14.4) to find the decoding metric for each message. If we then extend the node with greatest metric and continue this process until we reach a terminal node in the full tree, we can be reasonably sure that we will have found the same path that a MLD would have found with its enormously greater searching. This is the basic concept behind **sequential decoding**, the two principal forms of which will be described in the next two sections.

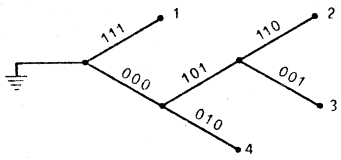

Fig. 14.1 A Partially-Explored $R = 1/3$ Tree Code for a Binary Input Channel.

Before launching into our discussion of sequential decoding, let us examine more closely the metric of (14.4) for a partially-explored tree code. If the node s is at depth u in the trellis, then $N_s = Nu$ (where N is the number of channel digits per branch) and the probability that the encoder reached this node in the tree in $P_s = 2^{-NRu}$. Substituting these values into (14.4) we find

$$L_f(s,\mathbf{y}) = \sum_{n=1}^{N_s} \left[\log \frac{P(y_n \mid x_{sn})}{P_Q(y_n)} - R\right] \tag{14.5}$$

which is the appropriate decoding metric for tree codes.

The metric (14.5) was first used for tree codes by Fano [14] (in whose honor we have used the subscript f.) It is a remarkable tribute to Fano's intuition that he postulated this metric entirely on intuitive grounds. The analytical justification given here was formulated by this writer [15] almost ten years later!

A word should be said about the probability distribution Q(x) which should be used in (14.2). That this should be the minimizing distribution in (2.2) can be seen as follows. If our tree code is to give a decoding error probability P_e with MLD bounded as in (3.5), it should have the character of a tree code whose digits were selected independently according to this Q(x). Given any partially explored section, we should presume that the further digits in the unexplored section have the character as though they were selected independently according to Q(x) and this is the only knowledge we should presume of these digits until we explore further into the tree.

A numerical example may help the reader to obtain a better "feel" for the Fano metric. Consider a tree code for a BSC with crossover probability $\epsilon = .045$. For any binary input channel, and thus in particular for the BSC, the minimizing distribution in (2.2) is $Q(0) = Q(1) = 1/2$. For $\epsilon = .045$, one finds from (2.2) that $R_0 = .50$ so it is natural in our example to choose a tree code of rate $R = 1/2$. From (14.5) we see that the Fano metric for the n-th digit of the s path in the tree is given by

$$L_f(x_{sn}) = \log \frac{1 - .045}{.050} - .50 \cong + .50$$

when $x_{sn} = y_n$ whereas

$$L_f(x_{sn}) = \log \frac{.045}{.50} - .50 \cong - 3.5$$

when $x_{sn} \neq y_n$. In practice, one usually scales the metrics by a positive constant so that all metric values can be closely approximated by integers which then permits the use of integer arithmetic in the decoding apparatus. In the present example, one would choose a scale factor of 2 so that

$$L_f(x_{sn}) = + 1 \text{ when } x_{sn} = y_n$$

$$- 7 \text{ when } x_{sn} \neq y_n.$$

In the above example, we have changed our notation for the Fano metric to reflect the fact that the received vector **y** may be considered as fixed for a particular decoding situation and to reflect the fact that the Fano metric is an additive function of the digits along the assumed encoded path **x**. We shall continue

henceforth with this simplified notation and write for instance

$$L_f(\mathbf{x}_s) = \sum_{n=1}^{N_s} L_f(x_{sn}) \tag{14.6}$$

where

$$L_f(x_{sn}) = \log \frac{P(y_n \mid x_{sn})}{P_Q(y_n)} - R\,. \tag{14.7}$$

15. Sequential Decoding--The Stack Algorithm

Sequential decoding is a generic name for any decoding procedure for a tree code which searches for the likely transmitted path for a given received sequence $y_{[0\ L\ +\ T)}$ by successively exploring the encoded tree with the following constraints on the nature of the exploration :

(1) Any new nodes explored must be at the next depth beyond an already explored node, and

(2) No knowledge of the unexplored part of the tree is available except knowledge of the statistical distribution Q(x) which characterizes the ensemble of tree codes for which the code is use may be considered a "typical" member.

The discussion of the previous section has undoubtedly led the reader to deduce that the "obvious" sequential decoding algorithm is one which stores the Fano metric for the paths to all terminal nodes already explored and then extends to the next depth in the tree that path with the greatest Fano metric. This in fact is just the **stack algorithm** first considered by Zigangirov [16] and independently proposed later by Jelinek [17]. The adjective "stack" is used to indicate that the "natural" decoder operation is to store the previously explored nodes in a stack with decreasing metric down into the stack and, at each step, to extend the node at the top of the stack.

For $R = 1/N$ tree codes where there are only two branches diverging from each node of the tree, we can state the strict **stack algorithm** as below where $\mathbf{x}_0$ denotes the encoded branch for the information digit 0 and $\mathbf{x}_1$ denotes the encoded branch for the information digit 1 on the two branches leading from the path i being extended. Each entry in the stack consists of a pair $[\mathbf{i}\,,\, L_f(\mathbf{x})]$ where $\mathbf{i}$ is the path through the tree, $\mathbf{x}$ the encoded digits on that path, and $L_f(\mathbf{x})$ the Fano metric for

$\mathbf{x}$. The symbol Λ denotes the "empty string" which is the path to the root node of the tree. For $\mathbf{i} = \Lambda$, we have $\mathbf{x} = \Lambda$ and $L_f(\mathbf{x}) = 0$.

Step 0: Place $[\Lambda, 0]$ into the initially empty stack.

Step 1: Extend the top entry $[\mathbf{i}, L_f(\mathbf{x})]$ in the stack by forming

$$[\mathbf{i} * 0, L_f(\mathbf{x}) + L_f(\mathbf{x}_0)]$$

and

$$[\mathbf{i} * 1, L_f(\mathbf{x}) + L_f(\mathbf{x}_1)]$$

then deleting $[\mathbf{i}, L_f(\mathbf{x})]$ from the stack.

Step 2: Place the two newly-formed entries into the stack so that the stack remains ordered with entries with greater metric higher in the stack.

Step 3: If the top entry $[\mathbf{i}, L_f(\mathbf{x})]$ in the stack is a path through the entire tree, stop and choose $\hat{\mathbf{i}}_{[0, L+T)} = \mathbf{i}$. Otherwise, go to step 1.

It should be very plausible to the reader, in light of the discussion in Section 14, that the stack algorithm does essentially maximum likelihood decoding (MLD). In fact, one can show quite precisely that P_e for stack decoding of the ensemble of random tree codes satisfies the bound (2.2) except for a somewhat greater value of the "unimportant" constant c_t [17].

As it stands, however, the "strict" stack algorithm is not a very practical decoding procedure because of the increasing time required for step 2 as the size of the stack grows. Jelinek in his proposal of the stack algorithm also suggested a clever technique to speed up this searching step, at negligible cost in increased probability of error, by ignoring metric differences within a small specified **quantization parameter** Δ. More precisely, he considers a stack of "buckets" $\ldots B_{-2}, B_{-1}, B_0, B_1, \ldots$ such that B_j contains all entries $[i, L(x)]$ in the stack for which

$$j\Delta \leqslant L_f(\mathbf{x}) < (j + 1)\Delta \tag{15.1}$$

and all "stores" and "fetches" from buckets are done on a "last in, first out" basis. Jelinek's **bucket stack algorithm** can be stated as follows:

Step 0: Place $[\Lambda, 0]$ into bucket B_0 of the initially empty stack.

Step 1: Fetch the most recent entry $[\mathbf{i}, L_f(\mathbf{x})]$ from the topmost non-empty bucket and form

$$[\mathbf{i} * 0, L_f(\mathbf{x}) + L_f(\mathbf{x}_0)]$$

and

$$[\mathbf{i} * 1, L_f(\mathbf{x}) + L_f(\mathbf{x}_1)].$$

Step 2: Store the two newly-formed entries into their appropriate buckets (as determined by (15.1).)

Step 3: If the most recent entry $[\mathbf{i}, L_f(\mathbf{x})]$ in the topmost non-empty bucket is a path through the entire tree, stop and choose $\hat{\mathbf{i}}_{[0,L+T)} = \mathbf{i}$. Otherwise, go to step 1.

Unlike the case for step 2 of the "strict" stack algorithm, the time required to execute step 2 of the "bucket" stack algorithm is independent of the number of entries already in the stack.

The practicality of the stack algorithm hinges on the amount of "computation" it requires before reaching a decoding decision. A computation is defined as the extension of one node, i.e. the performing of steps 1, 2 and 3 of the algorithm. Note that since one entry is deleted from the stack and two new entries are added in each computation, the number of computations performed at any time is equal to the number of entries in the stack. The quantity usually used to describe the computational effort is C_0, the number of computations required to decode divided by the number L of information bits decoded, i.e. the **per digit computation.** C_0 is of course a random variable whose value depends on how "noisy" is the received sequence $\mathbf{y}_{[0,L+T)}$. C_0 does not depend on the memory M of the convolutional code that might be used to generate the tree and depends very little on T. Hence, one usually chooses M and T so that the decoding error probability is negligibly small when the stack algorithm (or any other form of sequential decoding) is used. What limits performance is the decoding time available. If this time permits at most n_{max} computations, then the **deletion probability**

$$P_d = P(C_0 > \frac{n_{max}}{L}) \tag{15.2}$$

becomes the significant practical limitation. It turns out that C_0 is a particularly nasty kind of random variable, called a Pareto random variable (see [1] and [18]), for which $P(C_0 > n)$ decreases only as a small negative power of n. For this reason, P_d cannot be made extremely small with sequential decoding, $P_d \cong 10^{-3}$ being common in practice. Sequential decoding thus recommends itself to users who (1) have a feedback channel available on which to request repeats of deleted data, or (2) who do not mind deletion of small amounts of data, but who in either case demand a very small error probability in the undeleted data provided by the decoder.

The stack algorithm requires a substantial amount of storage but very little logical processing of its contents. Hence it is very well-matched to the

characteristics of present-day mini-computers.

It is interesting to consider the nature of the path that will be selected by the stack algorithm (provided that it is granted sufficient time to complete its computation.) Let V_j , $0 \leqslant j \leqslant L+T$, denote the metric along a given path **i** out to depth j in the tree, that is

$$V_j = L_f(\mathbf{x}_{[0,j)}) \tag{15.3}$$

where $\mathbf{x}_{[0,j)}$ is the encoded path for $\mathbf{i}_{[0,j)}$ and similarly let V'_j denote the metric out of depth j along another path **i**′. We can then state:

The Non-Selection Principle for the Stack Algorithm: If the paths **i** and **i**′ through the tree diverge at depth j and

$$\min(V_{j+1}, V_{j+2}, \dots V_{L+T}) > \min(V'_{j+1}, V'_{j+2}, \dots V'_{L+T}) \tag{15.4}$$

then **i**′ cannot be the path at the top of the stack when the stack algorithm stops.

To prove this property, we first note that since **i** and **i**′ diverge at depth j, we can write $\mathbf{i} = \mathbf{i}_{[0,j)} * \mathbf{i}_{[j,L+T)}$ and $\mathbf{i}' = \mathbf{i}_{[0,j)} * \mathbf{i}_{[j,L+T)}$. Hence, neither **i** nor **i**′ can be the final path unless at some point the entry $[\mathbf{i}_{[0,j)}, V_j]$ reaches the top of the stack and is extended so that both $[\mathbf{i}_{[0,j+1)}, V_{j+1}]$ and $[\mathbf{i}'_{[0,j+1)}, V'_{j+1}]$ then enter the stack. At every subsequent point, $\mathbf{i}_{[0,j+1)}$ or one of its extensions must be in the stack so there will always be an entry in the stack whose metric V satisfies

$$V \geqslant \min(V_{j+1}, V_{j+2}, \dots V_{L+T}). \tag{15.5}$$

Now suppose that V'_{j+k} is minimum among $V'_{j+1}, \dots V'_{L+T}$. If **i**′ reaches the top of the stack, then $\mathbf{i}'_{[0,j+k)}$ must have earlier reached the top of the stack since the complete path **i**′ is an extension of this path. But by (15.4) and (15.5), $V'_{j+k} < V$ so that $\mathbf{i}'_{[0,j+k)}$ cannot reach the top of the stack as there is always in the stack some entry with a greater metric. By contradiction then, we conclude that **i**′ cannot be the selected path.

Just as with the non-optimality principle for MLD, we can use the non-selection principle for stack decoding to find the path through the tree that will be found by a stack decoder. We would start at depth L - 1 and eliminate at each node that one of the 2 paths stemming therefrom with the smaller $\min(V_L, V_{L+1}, \dots V_{L+T})$. Then we would move to each node at depth L - 2

and eliminate that one of the two remaining paths stemming therefrom with the smaller $\min(V_{L-1}, V_L, \ldots V_{L+T})$, etc. When several centuries later (for L = 50, say) we reached depth 0, our elimination would leave us with one path, namely that one which would be selected finally by the stack algorithm. On the other hand, we might just apply the stack algorithm directly to the tree, starting at the root, and let it find its own path! The point is that (15.4) is only of importance conceptually in describing the nature of the finally selected path. Unlike the principal of non-optimality, it does not suggest directly a practical algorithm for finding the path of this nature.

It should be clear from the non-selection principle that P_e will be small for a stack decoder when and only when the metric tends to increase along the actual transmitted path through the tree but tends to decrease along every path diverging from this correct path. In fact, such reasoning is precisely what led Fano to the metric of (14.5). Fano reasoned that the first term in this metric is the "mutual information" between the received digit y_n and the hypothesized digit x_{sn}. If x_{sn} were the actual sent digit, the average of this mutual information should be channel capacity C so that the metric should on the average increase by the positive increment C - R at each digit on the correct path. On an incorrect path, the digit x_{sn} should be statistically independent of y_n so that the average mutual information should be 0 and thus the metric should on the average decrease by the negative increment - R at each digit along an incorrect path.

16. Sequential Decoding--The Fano Algorithm

The stack algorithm requires a substantial amount of storage in most applications. It uses this storage in order to perform an exploration of the tree that is "efficient" in the sense that each explored node is processed or "extended" only one time. The sequential decoding algorithm proposed by Fano [14] is a clever method for finding the path through the tree, as determined by the non-selection principle of the previous section, using almost no storage. The "trick" is to permit several extensions of the same node during the course of the search with all the necessary knowledge of previous searches reduced to a single stored number.

Perhaps Fano's algorithm can best be understood by an analogy. Consider the map of Fig. 16.1 which shows a tree of roads emanating from City A which we can consider the "root node" of the tree. (The "mysterious" city X whose elevation is very far below sea level is an artifice whose purpose will become clear but should

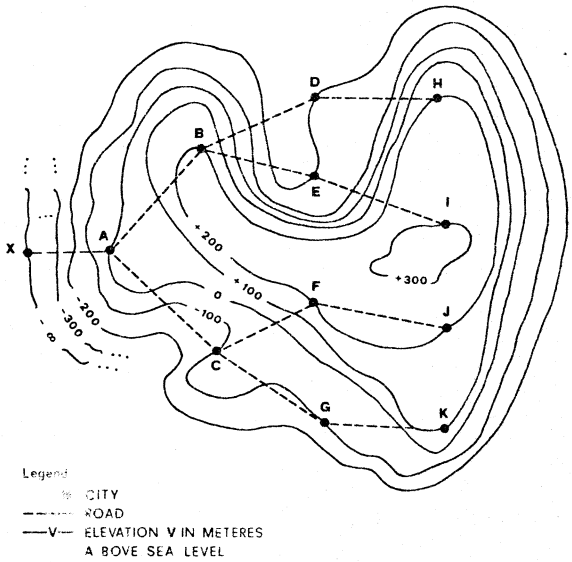

Fig. 16.1 Contour Map Used to Illustrate the Tree Search Performed by the Fano Algorithm.

be considered at depth - 1 from the root node A.) Suppose a traveller wishes to travel from A to that one of the terminal cities H, I, J, K that would be found by the stack algorithm, i.e. this path V_0, V_1, V_2, V_3 will win the inequality (15.4) against every path diverging from it. By applying the stack algorithm of the preceeding section, we find the stack contents after each computation to be as follows:

Computation No.	1	2	3	4	5
Contents of Stack	[A, 0]	[B, 200]	[C, –100]	[F, 200]	[J, 200]
		[C, –100]	[D, –200]	[G, –100]	[G, –100]
			[E, –200]	[D, –200]	[D, –200]
				[E, –200]	[E, –200]

Since J is a terminal city and is at the top of the stack, city J is the one that will be reached by a traveller who moves according to the stack algorithm. (J is not the highest terminal city, however; city I would be the one reached by a traveller who moved according to a MLD algorithm.) We see that it is necessary for the traveller to "remember" the elevation V of every city at the end of each of his explored paths in the course of his travelling in order to use the stack algorithm. On the other hand, the inequality (15.4) suggests that the traveller ought to be able, at each junction, to

choose the road toward the better city so long as this remains an "acceptable" path.

Suppose another traveller decides to use a "treshold of acceptability" T to decide when the path he is moving along has become unsatisfactory according to (15.4) so that he must turn back to try another path. He begins at A with $T = V_A = 0$. He sees that the elevation of the better city ahead is $V_B = 200$ which meets his threshold test so he moves to B. He now increases his threshold T to $T = V_B = 200$ since he hopes never to go lower. He now looks ahead but sees that neither city ahead meets his threshold. He then looks backwards and sees that this city A is also below his threshold. He has no recourse but to lower his threshold T to $T = 100$. Again he looks forward to D and E and sees that they violate his threshold, then he again looks back to A and sees that it fails his threshold test also. Again, he must lower his threshold T, this time to $T = 0$. Again he looks forward to D and E but again he cannot move. He then looks back to A and sees now that he can move **backward** to A which he does. As he moves back to A, he sees that he is on the better path from A so his first action at A is to look forward on the other path to C but he then sees that he cannot move there. He then looks back to the mysterious city X and sees that he cannot move there. (He is trapped, there is no path through the tree from A that stays at 0 or above at all points!) He then reduces his threshold T to $T = -100$ and again looks forward. Again he moves to the better city B; but what should he now do to his threshold? (The traveller has forgotten by now, but we know that if he raises his threshold again to $V_B = 200$ he will be travelling forever in a loop.) The traveller knows however that **the threshold was not tight** at A on this move and he uses this as a signal not to meddle with T which he leaves at $T = -100$. He then looks forward and sees he cannot move. He looks back and sees that he can move backward to A which he does. Seeing that he has returned by the better road, he then looks forward along the worse road to C. He sees that he can move to C which he does and since his threshold $T = -100$ is already tight he need not worry about changing it. He then looks forward and sees that he can move which he does along the better path to F. He recalls that his threshold was tight at C for this move so he thus raises his threshold now to $T = V_F = 200$. He now looks forward and sees that he can move to J which he does. Since J is a terminal city he has completed his journey and ended up in the same city as the traveller who used the stack algorithm.

This second traveller has made use of the **Fano Algorithm** to guide his search of the tree [14]. We give a flowchart for this algorithm in Fig. 16.2. We

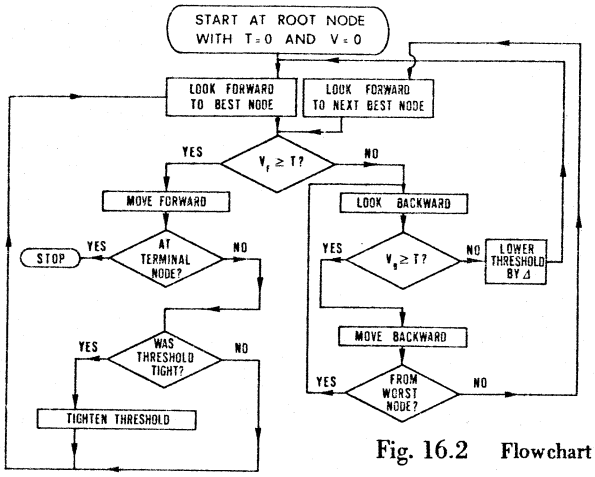

Fig. 16.2 **Flowchart for the Fano Sequential Decoding Algorithm.**

assume that there is a fictitious node, backwards from the root node, with metric $-\infty$ so that a look backward from the root node always results in lowering of the threshold by Δ. In this flowchart, V_f denotes the value of the metric for the path from the root node to the forward node at which one is looking, whereas V_B denotes the value of the metric for the path from the root node to the node being looked at which is behind the currently occupied node. By "tightening the threshold," we mean increasing T by the largest multiple of Δ such that the metric V from the root node to the currently occupied node still does not violate the threshold.

It is not hard to show that the Fano algorithm always finds the same path through the tree as the stack algorithm when (1) all node values are multiples of Δ and (2) there is no "tie" for the winning path, i.e. only one path through the tree is strictly not eliminated by the non-selection rule for the stack algorithm, cf. Geist [19]. Our traveller example suggests the key. The threshold will be lowered below 0 only at the root node and then only enough (if at all) to allow the path with greatest $\min(V_1, V_2, \ldots, V_{L+T})$ to be travelled. On the first arrival at any other node, the threshold will be tight before the next "look" since it was either tight on arrival or would be tightened by the outcome of the "was threshold tight" test. Hence, any other node has the same character search performed starting at first arrival there as does the root node. By iterating the root node argument, one finds that the path selected cannot be one that would have been eliminted by the non-selection rule. Since there is only one such path by assumption, one has verified that the Fano

algorithm will find the same path as the stack algorithm. When condition (1) is not met, then, as might be expected, the Fano algorithm will find the same path as the bucket stack algorithm for the same Δ provided again that there are no "ties" for the winning path, again cf. Geist [19].

Geist has also given some excellent suggestions for programming both the stack algorithm and the Fano algorithm [20]. One programming "trick" that should always be used since it substantially speeds up the computation is to use the difference $Q = V - T$ and to do all tests in terms of Q. In fact, one need not store V and T separately.

When Δ is properly chosen, the Fano algorithm is a very effective sequential decoding procedure. When both are programmed on a general-purpose computer, simulations suggest that the Fano algorithm will decode faster than the stack algorithm for $R < .9\, R_0$ [20]. Of course, if the available computer memory is small, one would be forced to use the Fano algorithm in any case.

It is curious to note the history of sequential decoding. The order of topics in Sections 14, 15 and 16 (which seems the most logical ordering) is actually in the reverse order of their discovery! The first sequential decoding algorithm, due to Wozencraft [6], inspired all these subsequent discoveries but is much more complicated as well as less effective than the stack and Fano algorithms. It has taken a long time to reach the point where one understands why the early sequential decoding algrithms "worked" and what they were really doing.

17. Aknowledgment and References

It would have been impossible to have reached the view of non-block codes that we have shared with the reader of these lectures without the stimulation we have received from engaging in search in this subject with our students. In particular, we must acknowledge Dr. Daniel J. Costello and Dr. John M. Geist, both of whom I had the pleasure to supervise as graduate students, and Mr. Rolf Johannesson who is just completing a year of study with me on leave from his home university, the Technical University of Lund, Sweden. Among colleagues who have greatly influenced my thoughts on non-block codes are, preeminently, Dr. G. David Forney, Jr., of the Codex Corporation, Newton, Mass., U.S.A., and Prof. Robert G. Gallager of M.I.T., Cambridge, Mass., U.S.A. Not least, I must mention the skillful introduction to, and encouragement to work in, the field of non-block coding that I received as a graduate student at M.I.T. in 1961-1962 under the supervision of Prof.

John M. Wozencraft.

Finally, I wish to thank Prof. Giuseppe Longo of the University of Trieste, Italy, for his invitation to present these lectures at the Centre International des Sciences Mécaniques, Udine, Italy, and for his interest in and encouragement of their written form.

REFERENCES

[1] Gallager, R.G., *Information Theory and Reliable Communication,* John Wiley and Sons, New York, 1968.

[2] Johannesson, R., "On The Error Probability of General Tree and Trellis Codes with Applications to Sequential Decoding," Tech. Rpt. No. EE 7316, Dept. of Elec. Engr., Univ. of Notre Dame, Notre Dame, Indiana, U.S.A. December 1973.

[3] Viterbi, A.J., "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Comm Tech.*, Vol. COM-19, pp. 751-772, October, 1971.

[4] Larsen, K.J., "Short Convolutional Codes with Maximum Free Distance for Rates 1/2, 1/3, and 1/4", *IEEE Trans. Info. Th.*, Vol. IT-19, pp. 371-372, May, 1973.

[5] Johannesson, R., "Robustly-Optimal Rate One-Half Binary Convolutional Codes", *IEEE Trans. Info. Th.*, Vol. IT-21, pp. 464-468, July 1975.

[6] Wozencraft, J.M., and B. Reiffen, *Sequential Decoding,* M.I.T. Press, Cambridge, Mass., 1959.

[7] Massey, J.L., *Threshold Decoding,* M.I.T. Press, Cambridge, Mass., 1963.

[8] Costello, D.J., Jr., "A Construction Technique for Random-Error-Correcting Convolutional Codes," *IEEE Trans. Info. Th.*, Vol. IT-15, pp. 631-636, September 1969.

[9] Forney, G. David, Jr., "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Info. Th.*, Vol. IT-16, pp. 720-738, November, 1970.

[10] Massey J.L., and M.K. Sain, "Inverses of Linear Sequential Circuits," *IEEE Trans. Computers,* Vol. C-17, pp. 330-337, April, 1968.

[11] Olson, R.R., "Note on Feedforward Inverses of Linear Sequential Circuits," *IEEE Trans. Computers,* Vol. C-19, pp. 1216-1221, December, 1970.

[12] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Info. Th.*, Vol. IT-13, pp. 260-269, April, 1967.

[13] Omura, J.K., "On the Viterbi Decoding Algorithm," *IEEE Trans. Info. Th.*, Vol. IT-15, pp. 177-179, January, 1969.

[14] Fano, R.M., "A Heuristic Discussion of Probabilistic Decoding," *IEEE Trans. Info. Th.*, Vol. IT-9, pp. 64-74, April 1963.

[15] Massey, J.L., "Variable-Length Codes and the Fano Metric," *IEEE Trans. Info. Th.*, Vol. IT-18, pp. 196-198, January, 1972.

[16] Zigangirov, K.Sh., "Some Sequential Decoding Procedures," *Prob. Pederachi Inform.*, Vol. 2. pp. 13-25, 1966.

[17] Jelinek, F., "A Fast Sequential Decoding Algorithm Using a Stack," *IBM J. Res. Dev.*, Vol. 13, pp. 675-685, November, 1969.

[18] Wozencraft, J.M., and I.M. Jacobs, *Principles of Communication Engineering*, John Wiley and Sons, New York, 1965.

[19] Geist, J.M., "Search Properties of Some Sequential Decoding Algorithms," *IEEE Trans. Info. Th.*, Vol IT-19, pp. 519-526, July, 1973.

[20] Geist, J.M., "A, Empirical Comparison of Two Sequential Decoding Algorithms," *IEEE Trans. Comm. Techn.*, Vol. COM-19, pp. 415-419, August, 1971.

[21] Van De Meeberg, L., " A Tightened Upper Bound on the Error Probability of Binary Convolutional Codes with Viterbi Decoding," *IEEE Trans. Info. Th.*, Vol. IT-20, pp. 389-391, May, 1974.