TDA Progress Report 42-100

February 15, 1990

Finding the Complete Path and Weight Enumerators of Convolutional Codes

I. Onyszchuk[1]
Communications Systems Research Section

A method for obtaining the complete path enumerator $T(D, L, I)$ of a convolutional code is described. A system of algebraic equations is solved, using a new algorithm for computing determinants, to obtain $T(D, L, I)$ for the (7,1/2) NASA standard code. Generating functions, derived from $T(D, L, I)$, are used to upper bound Viterbi decoder error rates. This technique is currently feasible for constraint length $K < 10$ codes.

A practical, fast algorithm is presented for computing the leading nonzero coefficients of the generating functions used to bound the performance of constraint length $K < 20$ codes. Code profiles with about 50 nonzero coefficients are obtained with this algorithm for the experimental $K = 15$, rate 1/4, code in the Galileo mission and for the proposed $K = 15$, rate 1/6, "2-dB" code.

I. Introduction

Convolutional codes such as the (7,1/2) NASA standard have been used for satellite and deep-space communications during the past 20 years. In 1971, Viterbi [1] defined generating functions for upper bounding error probabilities of convolutional codes on memoryless channels. In practice, for codes with more than eight states, these functions are still unknown, so error bounds have been evaluated using numerical matrix multiplications [2,8], which require extensive computations for each channel noise level. As an alternative, algorithms have also been developed to calculate the first few coefficients (the distance and bit error profiles) of these enigmatic generating functions [4,5]. Unfortunately, lists of these numbers are bulky (see Tables 2 and 3), and the minimum number of terms required to approximate the decoder error bounds depends upon the code rate and channel noise level.

A code's complete path enumerator $T(D, L, I)$ contains the number of paths having identical triples: weight, length, number of input 1s. The least-magnitude pole of the weight enumerator $T(D) = T(D, 1, 1)$ determines the point at which the union bounds [1] diverge, while additional poles and residues yield the dominant terms in the

[1] Also a student in the Electrical Engineering Department, California Institute of Technology, Pasadena, California.

partial fraction expansion of $T(D)$. Concise, yet very accurate, analytic approximations to several generating functions' coefficients may be obtained from these dominant terms. Furthermore, the poles and residues of $T(D)$ may eventually help unlock the structure of convolutional codes.

The generating functions $T(D)$ and $T(D,L,I)$ for the (7,1/2) NASA standard code (among many others) were obtained with a simple determinant algorithm described in this article. In addition, a new algorithm is presented for computing the initial coefficients of $T(D)$, $\partial T(D,L,I)/\partial L$ and $\partial T(D,L,I)/\partial I$ at $L=I=1$. These numbers are used to upper bound a Viterbi decoder's event, node, bit, and symbol error rates. The techniques explained here may be adapted, with modifications for code nonlinearity and Euclidean instead of Hamming distances, to find weight enumerators of trellis codes [3,5].

II. The Complete Path Enumerator

A binary, rate k/n, feed-forward convolutional encoder is defined by kn binary generator polynomials $g_{ij}(x)$, each representing the transfer function from the ith input to the jth output (x is a delay operator). Let $\underline{g}_{ij}$ be the vector whose rth component is the coefficient of x^r in $g_{ij}(x)$. The encoder's memory is $m = \sum_{i=1}^{k} \max_j [\deg\ g_{ij}(x)]$, which for rate $1/n$ codes is $K-1$. If the ith encoder memory cell contains s_i then the encoder is in state $s = \sum_{i=1}^{m} s_i 2^{i-1}$. The encoder's state diagram is a directed graph whose edges (corresponding to branches in the associated trellis diagram) have labels $a_{i,j} = D^d L I^b$ if there is an edge from state j into state i while $a_{i,j} = 0$ otherwise. During a transition from state j into state i, b is the number of ones input to the encoder and the number of ones output by the encoder is $d = \sum_{h=1}^{n} \underline{g}_{1h} \cdot [2j + (i \bmod 2)]_2$, where $\cdot$ is a modulo-2 inner product and $[k]_2$ denotes the binary representation of the integer k.

Let X_s be the trivariate generating function of all *simple* paths: those from state 0 into state s via nonzero states. X_0 counts all simple paths into state 0, called *fundamental* paths. Note that X_s is indexed by the destination state (s) while the source state (0) is constant. Now define $\underline{\mathbf{A}}$ as the 2^m-1 by 2^m-1 *adjacency matrix* of the encoder graph with state 0 and its edges removed. The entry in row i and column j of $\underline{\mathbf{A}}$ is $a_{i,j}$, which is nonzero only if there is a directed edge connecting state j to state i (they are adjacent). Now the following set of linear equations is constructed:

$$\underline{\mathbf{A}}\,[X_1, X_2, \ldots, X_{2^m-1}]^{\mathrm{T}} = [a_{1,0}, a_{2,0}, \ldots, a_{2^m-1,0}]^{\mathrm{T}}$$

By Cramer's Rule,

$$X_i = \frac{\det(\underline{\mathbf{A}}_i)}{\det(\underline{\mathbf{A}})}$$

for $i > 0$, where $\underline{\mathbf{A}}_i$ is $\underline{\mathbf{A}}$ with all column i entries $a_{r,i}$ replaced by $a_{r,0}$ for all rows r. A code's *complete path enumerator* (by weight, length, and number of input 1s)

$$T(D,L,I) = X_0 = \sum_{j=1}^{2^k} a_{0,z_j} X_{z_j}$$

where $\{z_j\}$ are the 2^k states having edges into state 0 (so $a_{0,z_j} \neq 0$).

As an example with $k=1$ for simplicity, the $m=2$ encoder in Fig. 1 is in state $s = 2s_1 + s_0$. The corresponding state diagram in Fig. 2 leads to the equations $\underline{\mathbf{A}}\,[X_1, X_2, X_3]^{\mathrm{T}} = [a_{1,0}, 0, 0]^{\mathrm{T}}$:

$$\begin{bmatrix} 1 & -LI & 0 \\ -DL & 1 & -DL \\ -DLI & 0 & 1-DLI \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} D^2LI \\ 0 \\ 0 \end{bmatrix}$$

Now $T(D,L,I) = X_0 = D^2 L X_2 = D^5 L^3 I/(1-DLI-DL^2I)$, so the code's weight enumerator is

$$T(D) = \frac{D^5}{1-2D} = \sum_{d=5}^{\infty} 2^{d-5} D^d = \sum_{d=d_{\mathrm{free}}}^{\infty} p(d) D^d$$

where $p(d) = 2^{d-5}$ is the number of weight d fundamental paths, and $d_{\mathrm{free}} = 5$ is the code's free distance.

III. Reducing an Adjacency Matrix

When $k=1$, there are only $3(2^m-1)-2$ nonzero out of $(2^m-1)^2$ entries in $\underline{\mathbf{A}}$, and they are located in a special pattern. Since $\underline{\mathbf{A}}$ is sparse if $m > 3$, reduction is useful before computing the determinant. The following example, different from the one in the previous section, illustrates the reduction procedure. The code with generator polynomials $g_{11}(D) = 1+D+D^3$ and $g_{12}(D) = 1+D+D^2+D^3$

has an adjacency matrix $\underline{\mathbf{A}}$ (with $L = I = 1$ to simplify entries)

$$\begin{bmatrix} 1 & 0 & 0 & (D^2)-1 & 0 & 0 & 0 \\ -D^2 & 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & -D^2 & 0 & 0 \\ 0 & -D & 0 & (0)1 & 0 & -D & 0 \\ 0 & -D & 0 & 0 & 1 & -D & 0 \\ 0 & 0 & -D & 0 & 0 & 1 & -D \\ 0 & 0 & -D & 0 & 0 & 0 & 1-D \end{bmatrix}$$

The round brackets (parentheses) above indicate values in $\underline{\mathbf{A}}_{2^{m-1}}$ that are different from those in $\underline{\mathbf{A}}$. Since $k = 1$, the determinants of only these two matrices are needed for $T(D, L, I)$, and this notation will lead to their simultaneous computation.

As in Gaussian elimination, $-a_{r,\lfloor r/2 \rfloor}$ times row $\lfloor r/2 \rfloor$ is added to rows $r = 2$ to 2^m-1 so that $\underline{\mathbf{A}}$ becomes zeroed below its main diagonal for columns 1 to $2^{m-1}-1$:

$$\begin{bmatrix} 1 & 0 & 0 & (D^2)\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & (D^4)-D^2 & -1 & 0 & 0 \\ 0 & 0 & 1 & (D^2)\ -1 & -D^2 & 0 & 0 \\ 0 & 0 & 0 & (D^5)1-D^3 & -D & -D & 0 \\ 0 & 0 & 0 & (D^5)-D^3 & 1-D & -D & 0 \\ 0 & 0 & 0 & (D^3)-D & -D^3 & 1 & -D \\ 0 & 0 & 0 & (D^3)-D & -D^3 & 0 & 1-D \end{bmatrix}$$

Therefore, $\det(\underline{\mathbf{A}})$ equals the determinant of the resulting lower right 2^{m-1} by 2^{m-1} submatrix. To further reduce $\underline{\mathbf{A}}$, row $\lfloor r/2 \rfloor$ times $-a_{r,2^{m-1}+\lfloor r/2 \rfloor}$ is added to each row $r = 2^m-3$ to 2^{m-1} (4 to 5 here), so that columns $2^{m-1}+2^{m-2}$ to 2^m-2 (both 6 here) are zeroed above the main diagonal. Now $\det(\underline{\mathbf{A}}) = \det(\underline{\tilde{\mathbf{A}}})$, where $\underline{\tilde{\mathbf{A}}}_{2^{m-1}})$ and $\underline{\tilde{\mathbf{A}}}$ are the new lower right 2^{m-1} by 2^{m-1} submatrices:

$$\begin{bmatrix} (D^4+D^5)1-D^2-D^3 & -D-D^4 & 0 & -D^2 \\ (D^4+D^5)-D^2-D^3 & 1-D-D^4 & 0 & -D^2 \\ (D^3)\ -D & -D^3 & 1 & -D \\ (D^3)\ -D & -D^3 & 0 & 1-D \end{bmatrix}$$

To zero column 2^{m-1} of $\underline{\tilde{\mathbf{A}}}$ (above the diagonal entry $\tilde{a}_{2^{m-1},2^{m-1}} = a_{2^{m-1},2^{m-1}} \neq 0$), $-\tilde{a}_{r,2^{m-1}}/\tilde{a}_{2^{m-1},2^{m-1}}$ times row 2^{m-1} is added to each row $r = 1$ to $2^{m-1}-1$. Define $(\underline{\mathbf{B}}_{2^{m-1}})$ and $\underline{\mathbf{B}}$ as the resulting upper left 2^{m-2} by 2^{m-2} submatrices,

$$\frac{\begin{bmatrix} (D^4+D^5-D^6)1-D-D^2-D^3+D^4 & -D+D^2-D^4 \\ (D^4+D^5-D^6) & -D^2-D^3+D^4 \quad 1-2D+D^2-D^4 \end{bmatrix}}{1-D}$$

This reduction method simultaneously produces two dense 2^{m-2} by 2^{m-2} matrices $(\underline{\mathbf{B}}_{2^{m-1}})$, $\underline{\mathbf{B}}$ with the same determinants as the corresponding original sparse 2^m-1 by 2^m-1 matrices $(\underline{\mathbf{A}}_{2^{m-1}})$, $\underline{\mathbf{A}}$.

IV. A Determinant Algorithm

The following algorithm yields the determinant of any $N \times N$ matrix $\underline{\mathbf{B}}$ having entries from a Euclidean domain, such as the set of all polynomials with integer coefficients. A sequence of matrices $\underline{\mathbf{B}}^{(\mathbf{N})}, \underline{\mathbf{B}}^{(\mathbf{N-1})}, \underline{\mathbf{B}}^{(\mathbf{N-2})}, \ldots, \underline{\mathbf{B}}^{(\mathbf{1})}$ is computed with each matrix having the same determinant (up to sign) as $\underline{\mathbf{B}}$. Starting with $\underline{\mathbf{B}}^{(\mathbf{N})} = \underline{\mathbf{B}}$, step j in the algorithm produces the numerator $b_{i,k}^{(j-1)}$ of each entry in row i and column k of $\underline{\mathbf{B}}^{(\mathbf{j-1})}$:

for $j = N$ **to** 2 (step)

for $i = j-1$ **to** 1 (row index)

for $k = j$ **to** 1 (column index)

$$b_{i,k}^{(j-1)} = \frac{b_{i,k}^{(j)}\, b_{j,j}^{(j)} - b_{i,j}^{(j)}\, b_{j,k}^{(j)}}{b_{j+1,j+1}^{(j+1)}}$$

Naturally, if $b_{j+1,j+1}^{(j+1)} = 0$ prior to step j, then any column k such that $b_{j+1,k}^{(j+1)} \neq 0$ must first be interchanged with column $j+1$. This operation negates the determinant, so a counter t is incremented to record the event. If no such column k exists, row $j+1$ is zero, so the algorithm is stopped and $\det(B) = \det(B^{(j)}) = 0$. Also note that $b_{N+1,N+1}^{(N+1)} = 1$ initially.

The following example with $N = 4$ illustrates the above reduction procedure. When $j = 4$, after $-b_{34}^{(4)}/b_{44}^{(4)}$, $-b_{24}^{(4)}/b_{44}^{(4)}$, and $-b_{14}^{(4)}/b_{44}^{(4)}$ times row 4 are added to rows 3, 2, and 1, respectively (corresponding to $j = N = 4$ in the

algorithm), in order to zero all entries in column 4 above $b_{44}^{(4)}$,

$$\underline{\mathbf{B}}^{(3)} = \begin{bmatrix} \frac{b_{11}^{(3)}}{b_{44}^{(4)}} & \frac{b_{12}^{(3)}}{b_{44}^{(4)}} & \frac{b_{13}^{(3)}}{b_{44}^{(4)}} & 0 \\ \frac{b_{21}^{(3)}}{b_{44}^{(4)}} & \frac{b_{22}^{(3)}}{b_{44}^{(4)}} & \frac{b_{23}^{(3)}}{b_{44}^{(4)}} & 0 \\ \frac{b_{31}^{(3)}}{b_{44}^{(4)}} & \frac{b_{32}^{(3)}}{b_{44}^{(4)}} & \frac{b_{33}^{(3)}}{b_{44}^{(4)}} & 0 \\ b_{41}^{(4)} & b_{42}^{(4)} & b_{43}^{(4)} & b_{44}^{(4)} \end{bmatrix}$$

Then, after the $j = 3$ step during which $-b_{23}^{(3)}/b_{33}^{(3)}$ and $-b_{13}^{(3)}/b_{33}^{(3)}$ times row 3 are added to rows 2 and 1,

$$\underline{\mathbf{B}}^{(2)} = \begin{bmatrix} \frac{b_{11}^{(2)}}{b_{33}^{(3)}} & \frac{b_{12}^{(2)}}{b_{33}^{(3)}} & 0 & 0 \\ \frac{b_{21}^{(2)}}{b_{33}^{(3)}} & \frac{b_{22}^{(2)}}{b_{33}^{(3)}} & 0 & 0 \\ \frac{b_{31}^{(3)}}{b_{44}^{(4)}} & \frac{b_{32}^{(3)}}{b_{44}^{(4)}} & \frac{b_{33}^{(3)}}{b_{44}^{(4)}} & 0 \\ b_{41}^{(4)} & b_{42}^{(4)} & b_{43}^{(4)} & b_{44}^{(4)} \end{bmatrix}$$

Since only elementary row operations on $\underline{\mathbf{B}}$ have been performed, $\det(\underline{\mathbf{B}}) = \det(\underline{\mathbf{B}}^{(3)}) = \det(\underline{\mathbf{B}}^{(2)})$. The final step ($j = 2$) produces a $\underline{\mathbf{B}}^{(1)}$ matrix identical to $\underline{\mathbf{B}}^{(2)}$ except with $b_{11}^{(1)}/b_{22}^{(2)}$ 0 0 0 in the first row. Therefore, $\det(\underline{\mathbf{B}}) = \det(\underline{\mathbf{B}}^{(1)}) = b_{11}^{(1)}$. This result generalizes to any nonsingular $N \times N$ matrix $\underline{\mathbf{B}}$.

Lemma. Entry $b_{1,1}^{(1)} = \det(\underline{\mathbf{B}}^{(1)}) = (-1)^t \det(\underline{\mathbf{B}})$, where t is the total number of column interchanges performed by the algorithm in computing $b_{1,1}^{(1)}$.

Proof. For each value of j from N to 2, when $k = j$, the algorithm makes $b_{i,j}^{(j-1)} = 0$ for all $1 \le i \le j-1$ so that all column j entries above the main diagonal become 0. Since $\underline{\mathbf{B}}^{(1)}$ is zero above its main diagonal, $\det(\underline{\mathbf{B}})$ is the product of $(-1)^t$ and the diagonal entries in $\underline{\mathbf{B}}^{(1)}$.

$$\det(\underline{\mathbf{B}}) = (-1)^t \det(\underline{\mathbf{B}}^{(1)})$$

$$= (-1)^t \prod_{j=1}^{N} \frac{b_{j,j}^{(j)}}{b_{j+1,j+1}^{(j+1)}}$$

$$= (-1)^t \, b_{1,1}^{(1)} \quad \bullet$$

Expanding $b_{i,k}^{(j)} \, b_{j,j}^{(j)} - b_{i,j}^{(j)} \, b_{j,k}^{(j)}$ using the equation in the algorithm shows that $b_{j+1,j+1}^{(j+1)}$ divides this expression so there are never any remainders. This is expected because the algorithm implements a recursive factorization of the determinant written as a sum of products of matrix entries. The algorithm differs from standard Gaussian elimination because the diagonal entries in the reduced matrices are not made equal to 1. Also, after calculations with a particular value of j are completed, all entries' denominators are previous pivot numerators.

V. Path Distance, Length, and Bit Error Approximations

The complete path enumerator $T(D, L, I)$ for the $m = 6$, rate 1/2, NASA standard code was obtained by using the preceding algorithm to simultaneously compute determinants of the two 16 by 16 reduced matrices $\underline{B}_{2m-1}$ and $\underline{B}$. The 76 poles of $T(D)$ for this code are plotted on the complex plane (Fig. 3) along with the unit circle for reference. Using only the six least-magnitude poles (indicated by the large points in Fig. 3), an approximation to the partial fraction expansion of $T(D)$ is

$$T(D) \approx D^{10} \left[\frac{r_1}{1-\alpha_d D} + \frac{r_1}{1+\alpha_d D} + \frac{r_2}{1-\alpha_b} + \frac{r_2}{1+\alpha_b D} + \frac{r_2^*}{1-\alpha_b^* D} + \frac{r_2^*}{1+\alpha_b^* D} \right]$$

$$= D^{10} \sum_{k=0}^{\infty} \left[r_1(\alpha_d)^k + r_1(-\alpha_d)^k + r_2(\alpha_b)^k + r_2(-\alpha_b)^k + r_2^*(\alpha_b^*)^k + r_2^*(-\alpha_b^*)^k \right] D^k$$

$$= \sum_{j=0}^{\infty} \left[2r_1(\alpha_d)^{2k} + 4\mathrm{Re}\{r_2(\alpha_b)^{2k}\} \right] D^{2k+10}$$

where $\alpha_d = 2.3876225$ is the reciprocal of the least-magnitude pole of $T(D)$, $\alpha_b = 1.657193e^{-0.983418j}$ is the reciprocal of the pole with next smallest magnitude, * means complex conjugate, and $j = \sqrt{-1}$. The residues are

$$r_1 = \frac{-P(\alpha_d^{-1})}{\alpha_d^{-1} Q'(\alpha_d^{-1})} \quad \text{and} \quad r_2 = \frac{-P(\alpha_b^{-1})}{\alpha_b^{-1} Q'(\alpha_b^{-1})}$$

where $D^{10}P(D)/Q(D) = T(D)$ and $Q'(D)$ is the derivative of $Q(D)$. The other poles in Fig. 3 may be ignored because their magnitudes are greater than 0.8, and the corresponding residues have magnitudes less than 0.07.

Define $p(d), i(d)$, and $\ell(d)$ as the coefficients of D^d in $T(D), \partial T(D,L,I)/\partial L$, and $\partial T(D,L,I)/\partial L$, respectively, at $L = I = 1$ (these generating functions are shown in the Appendix). The number of fundamental paths having weight $2k + 10$ is

$$p(2k+10) \approx 6.82(2.3876225)^{2k} + 4.25(1.657193)^{2k}\cos(0.310 - 1.967k)$$

Similarly, the terms corresponding to the six least-magnitude poles of $T(D)$ in the partial fraction expansions of $\partial T(D,L,I)/\partial L$ and $\partial T(D,L,I)/\partial I$ at $L = I = 1$ were used to obtain the approximations

$$\ell(2k+10) \approx (77.725 + 22.625k)(2.3876225)^{2k} + 39.3(1.657193)^{2k}\cos(0.485 - 1.967k) + (2k+1)7.2676(1.657193)^{2k}\cos(0.383 - 1.967k)$$

$$i(2k+10) \approx (24.474 + 12.018k)(2.3876225)^{2k} + 9.942(1.657193)^{2k}\cos(0.575 - 1.967k) + (2k+1)2.8723(1.657193)^{2k}\cos(0.366 - 1.967k)$$

which have a relative error < 0.0001 for $k > 4$.

A rate $1/n$ Viterbi decoder's bit error rate (BER) on a binary-input, output-symmetric [2], discrete memoryless channel is bounded by

$$\text{BER} \le \sum_{d=d_{free}}^{\infty} i(d)P_d$$

where P_d is the probability that the decoder outputs a fundamental path having distance d from the one transmitted. The probability that a b-bit symbol is decoded incorrectly is bounded by [9]

$$\text{SER}_b \le \sum_{d=d_{free}}^{\infty} [[b - 1 - m]p(d) + \ell(d)]\, P_d$$

For the additive white Gaussian noise (AWGN) channel with bit signal-to-noise ratio E_b/N_0,

$$P_d = Q(\sqrt{2dRE_b/N_0})$$

where $Q(x)$ is the Gaussian integral function [2, p. 62]. On a binary symmetric channel with crossover probability p,

$$P_{2i} = P_{2i-1} = \sum_{e=i}^{2i-1} \binom{2i-1}{e} p^e (1-p)^{2i-1-e} \le \binom{2i-1}{i} p^i$$

[6]. For decoders using integer metrics, as for example on a binary-input, output-quantized AWGN channel, P_d can also be computed exactly [2, p. 291].

VI. Algorithms for Profiles of Convolutional Codes

Finding the complete path enumerator of codes with memory greater than 8 currently seems infeasible. In these cases, an algorithm for distance profiles [4,5] may be used to calculate the first few nonzero coefficients of the generating functions used for error bounds. However, these methods are fairly complex and some require extra computation to ensure that the output is correct. In this section, Viterbi's algorithm, with survivors replaced by vectors of integers that count paths, lengths, or bits, is applied on a noiseless channel to compute $p(d), \ell(d)$, and $b(d)$ values. Rate $1/n$ codes are treated first to simplify the discussion. Define $\text{out}_0[s]$ and $\text{out}_1[s]$ as the number of ones that the encoder outputs going from state $s0 = \lfloor s/2 \rfloor$ and $s1 = s0 + 2^{m-1}$ into state s. Analogous to a state metric, the entry in row $s > 0$ and column 0 of a matrix W, referred to as $W[s][0]$, will contain the *least weight* of any simple path with length $\le T$ trellis branches. For $t = 1$ to coeffs (a parameter described later), $W[s][t]$ will be the *number* of simple paths of weight $W[s][0]+t-1$ and length $\le T$ branches into state $s > 0$. For state 0, $W[0][t]$ is always kept at 0, except $W[0][1] = 1$. The entries in a second matrix B count either the total number of ones input to the encoder (when the variable len = 0) or the total length in trellis branches (when len = 1), of all simple paths having length $\le T$ (again $B[0][t] = 0$ always). These matrices are obtained for successive values of T starting with 1 by extending, one branch length at a time (an algorithm 'step'), the code trellis starting from state 0 only. Thus the longest pathlength (T) explored by the algorithm equals the number of 'steps' executed. The algorithm terminates after step T^* when W has reached values that will never change, which also forces B to remain constant.

Then since $W[s][0]$ is the least weight of any simple path into state $s > 0$, $d_{\text{free}} = W[2^{m-1}][0] + \text{out}_1[0]$. Also,

$$W[2^{m-1}][d - d_{\text{free}} + 1] = p(d)$$

$$B[2^{m-1}][d - d_{\text{free}} + 1] = b(d) \quad (\ell(d) \text{ if len} = 1)$$

for $d = d_{\text{free}}$ to $d_{\text{free}} + \text{coeffs} - 1$.

Two versions of the basic algorithm above are presented in C language format in Algorithm 1 and Table 1. In Algorithm 1, matrices P and A store *previous* W and B entries corresponding to simple paths of length $\leq T$, which are used to compute new W and B matrices for length $\leq T+1$ simple paths. When change remains 0 after step T^*, W (and thus B) will never change because $W[s][t] = P[s][t]$ for all s and t. $P[s][0]$ is initialized to 999 for $s > 0$, $P[0][1] = 1$, and all other array values are initialized to 0. If any second array index t + offset is ≤ 0 in the $W[s][t]$ and $B[s][t]$ instructions, the array referenced is simply ignored.

The algorithm requires storage for $2^{m+1}(\text{coeffs} + 2)$ integers and the amount of work per step is proportional to this number. The number of steps executed, T^*, equals the length, in trellis branches, of the longest fundamental path(s) having weight d_{free} + coeffs − 1. The parameter coeffs should be set equal to $\lceil 10n/k \rceil$ because using this many nonzero terms in the union bounds gives results with three significant digits of precision when the bounds are tight enough to be useful.

Setting coeffs = 0 and ignoring offset0, offset1, A, and B yields a simple and fast algorithm for finding d_{free}. About 2^{m+1} bytes of storage and 2.1 CPU seconds (on a computer executing 12 million instructions per second) were required to obtain d_{free} for the $m = 14$ Galileo and "2-dB" codes.

For rate k/n codes with $k > 1$, a state s is partitioned into $s_1, s_2, \ldots, s_k$ where s_i corresponds to the contents of the ith encoder shift register. The output weight as the encoder enters state s is $\text{out}_j[s]$ and the input is $j \in [0 \ldots 2^k - 1]$. New $W[s][t]$ and $B[s][t]$ values are computed using at most 2^k entries from each of the P and A matrices.

For codes with $2^k << 2^m$ (such as rate $1/n$ with $m > 3$), Algorithm 1 may be improved by looping through groups of 2^k states called butterflies (see Fig. 4) instead of individual states and by computing $W[s][t]$ and $B[s][t]$ in place [7]. This reduces the storage memory required by almost one-half because the "double-buffering" matrices P and A are eliminated. If the entire vector $W[2^{m-1}]$ remains the same for m consecutive steps, it will never change because there is a trellis path of length $\leq m$ branches between any two states. The algorithm shown in Table 1, which incorporates these improvements, produced the first $10n$ nonzero coeffs in the profiles of two $K = 15$ codes: the rate 1/4 Galileo code (Table 2) and the proposed "2-dB", rate 1/6 code (Table 3). These profiles took only a few minutes of CPU to generate and required storage of 7 and 8.5 Mbytes, respectively, when 4 bytes were used for each integer. These memory requirements could be further reduced by storing each integer in the smallest number of bytes needed (1 for $W[s][0]$ to $W[s][15]$ in the $K = 15$ codes above).

Algorithm 1. A simple profile algorithm

```
do {
    for (s = 1 to 2^m − 1) {
        s0 = ⌊s/2⌋; s1 = s0 + 2^(m−1);
        bit = len + (1 − len) * (s mod 2) ;
        W[s][0] = min (P[s0][0] + out0[s], P[s1][0] + out0[s]);
        offset0 = W[s][0] − P[s0][0] − out0[s];
        offset1 = W[s][0] − P[s1][0] − out0[s];
        for (t = 1 to coeffs) {
            W[s][t] = P[s0][t + offset0] + P[s1][t + offset1];
            B[s][t] = A[s0][t + offset0] +
                      A[s1][t + offset1] + bit * W[s][t]; }
    } change = 0;
    for (s = 1 to 2^m − 1)
        for (t = 0 to coeffs)
            if (P[s][t] ≠ W[s][t]) { change = 1;
                P[s][t] = W[s][t]; B[s][t] = A[s][t]; }
} while (change ≠ 0);
```

References

[1] A. J. Viterbi, "Convolutional Codes and their Performance in Communication Systems," *IEEE Trans. Comm.*, COM-19, pp. 751–772, October 1971.

[2] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding.* New York: McGraw-Hill, 1979.

[3] E. Zehavi and J.K. Wolf, "On the Performance Evaluation of Trellis Codes," *IEEE Trans. Info. Theory*, vol. IT-33, pp. 196-202, March 1987.

[4] L. Chevillat and R. Johannesson, "A Fast Algorithm for Finding the Distance Spectrum of Convolutional Codes," *IEEE Trans. Info. Theory*, to appear.

[5] M. Rouanne and D.J. Costello, Jr., "An Algorithm for Computing the Distance Spectrum of Trellis Codes," *IEEE Journal on Selected Areas of Comm.*, vol. SAC-21, August 1989.

[6] L. Van de Meeburg, "A Tightened Upper Bound on the Error Probability of Binary Convolutional Codes with Viterbi Decoding," *IEEE Trans. Info. Theory*, vol. IT-20, pp. 389-391, May 1974.

[7] C. M. Rader, "Memory Management in a Viterbi Decoder," *IEEE Trans. Comm.*, COM-29, pp. 1399-1401, September 1981.

[8] P. J. Lee, "A Very Efficient Transfer Function Bounding Technique on Bit Error Rate for Viterbi Decoded, Rate 1/N Convolutional Codes," *TDA Progress Report 42-79*, vol. July–September, pp. 114-123, November 15, 1984.

[9] R. J. McEliece and I. M. Onyszchuk, "A Symbol Error Upper Bound for Convolutional Codes," *Proc. 27th Allerton Conf.*, Monticello, Illinois, September 26–29, 1989.

Table 1. A profile algorithm for rate 1/n codes

```
k=0; stop = 0;
do {
    for (t = 0 to coeffs)
        { temp[t] = W[2^{m-1-k}][t]; B[0][t] = W[0][t] = 0; }
    W[0][1] = 1;
    for (s = 0 to 2^{m-1} - 1) {
        s0 = s >> k;    (cyclically)
        s1 = s0 + 2^{m-1-k}
        tw0[0] = min (W[s0][0] + out_0[2s], W[s1][0] + out_1[2s]);
        tw1[0] = min (W[s0][0] + out_1[2s+1], W[s1][0] + out_0[2s+1]);
        offset00 = tw0[0] - W[s0][0] - out_0[2s];
        offset01 = tw0[0] - W[s1][0] - out_1[2s];
        offset10 = tw1[0] - W[s0][0] - out_1[2s+1];
        offset11 = tw1[0] - W[s1][0] - out_0[2s+1];
        for (t = 1 to coeffs) {
            tw0[t] = W[s0][t + offset00] + W[s1][t + offset10];
            tw1[t] = W[s0][t + offset10] + W[s1][t + offset11];
            tb0[t] = B[s0][t + offset00] + B[s1][t + offset10] + len * tw0[t];
            tb1[t] = B[s0][t + offset10] + B[s1][t + offset11] + tw1[t];
        }
        for (t = 0 to coeffs) {
            W[s0][t] = tw0[t]; W[s1][t] = tw1[t];
            B[s0][t] = tb0[t]; B[s1][t] = tb1[t]; }
        k = k+1; if (k = m) k = 0; change = 1;
        if (W[2^{m-1-k}][coeffs] > 0) change = 0;
            for (t = 0 to coeffs)
                if (W[2^{m-1-k}][t] ≠ temp[t]) change = 1;
        if (change = 1) stop = 0;
        if (change = 0 and stop < m-1) {change = 1; stop++; }
} while (change ≠ 0);
```

Table 2. Galileo code profiles

Distance d	Fundamental paths $p(d)$	Bit errors $i(d)$	Total lengths $\ell(d)$
35	2	6	7
36	1	2	5
37	4	16	22
38	2	8	12
39	3	11	17
40	5	20	28
41	6	24	46
42	17	76	122
43	24	126	214
44	29	180	285
45	39	255	438
46	66	416	721
47	94	628	1071
48	121	850	1478
49	175	1313	2260
50	277	2086	3643
51	415	3361	5855
52	639	5304	9388
53	934	8010	14161
54	1273	11452	20271
55	1906	17550	31381
56	2878	27332	49172
57	4054	39750	71705
58	5978	60788	109808
59	8864	92738	167861
60	12966	139556	253134
61	18984	210112	383008
62	27949	317798	581467
63	41092	479512	878975
64	60126	720858	1323152
65	87799	1080933	1987235
66	128712	1622990	2992979
67	189880	2451782	4530508
68	278589	3682496	6817868
69	408780	5534126	10261968
70	598271	8283100	15386816
71	875283	12380669	23050515
72	1286052	18596544	34662286
73	1888299	27885609	52045238
74	2768375	41727376	78013493
75	4057688	62421220	116865844
76	5953416	93419654	175122289
77	8732134	139709066	262220198
78	12809968	208928290	392628663
79	18786484	312181796	587384902
80	27548175	466271448	878292728
81	40412499	696477455	1313354906
82	59269748	1039725314	1962719710

Table 3. Rate 1/6 "2-dB" code profiles

Distance d	Fundamental paths $p(d)$	Bit errors $i(d)$	Total lengths $\ell(d)$
56,57	1,5	2,15	3,19
58,60	1,3	2,12	2,14
61,62	5,12	25,56	35,84
63,64	11,5	43,24	67,40
65,66	8,11	44,62	68,95
67,68	8,11	48,62	76,98
69	27	167	267
70	30	162	277
71	36	216	363
72	54	366	573
73	74	464	785
74	89	610	998
75	94	670	1104
76	126	912	1524
77	163	1209	2022
78	226	1676	2814
79	290	2236	3785
80	369	2920	4993
81	493	4051	6846
82	574	4780	8168
83	767	6571	11236
84	979	8562	14687
85	1182	10474	18250
86	1574	14282	24860
87	1996	18516	32193
88	2618	24594	43183
89	3407	32955	57577
90	4238	41914	73499
91	5353	53757	94399
92	7006	71430	126401
93	8932	92712	164631
94	11418	120946	214330
95	14401	155175	275986
96	18467	202902	361135
97	24039	268439	479664
98	30325	344146	616671
99	38662	446878	800288
100	49690	583672	1048171
101	63930	762130	1371587
102	81742	990268	1785532
103	103839	1278325	2308219
104	133335	1666564	3012971
105	170357	2159215	3912282
106	217467	2801764	5081111
107	278512	3640320	6613934
108	356223	4721974	8592622
109	456347	6135943	11180051
110	583546	7956498	14517787
111	746528	10327464	18861798
112	954389	13376948	24468075
113	1220261	17333391	31748900
114	1562164	22493842	41242505
115	1997088	29126250	53477032

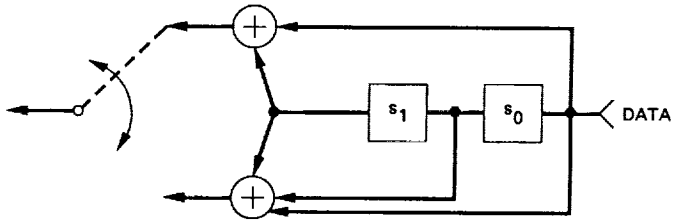

Fig. 1. A rate 1/2, 4-state encoder.

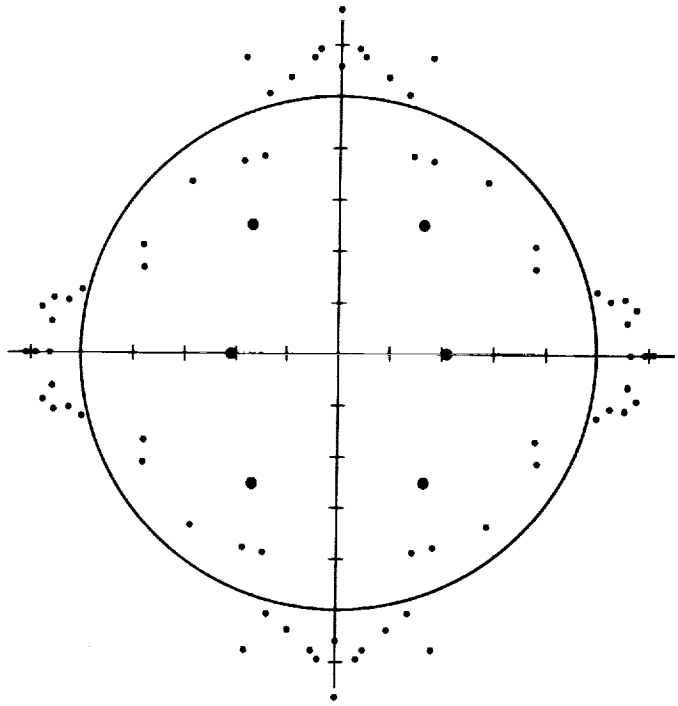

Fig. 3. Poles of $T(D)$ for the (7,1/2) NASA code.

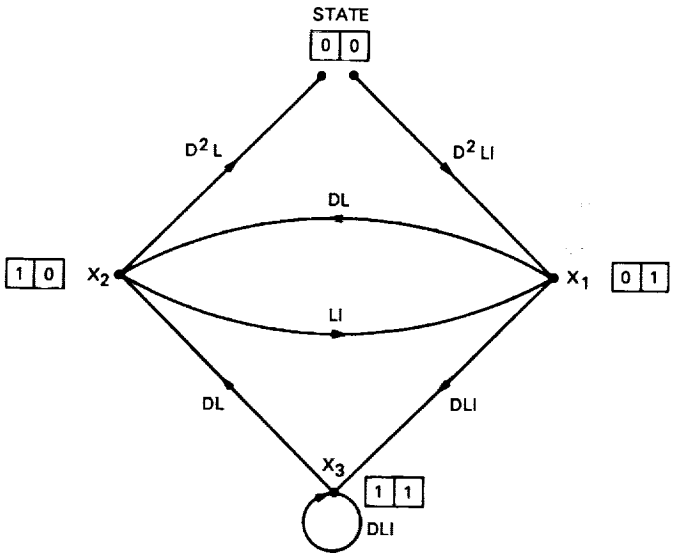

Fig. 2. State diagram of the encoder in Fig. 1.

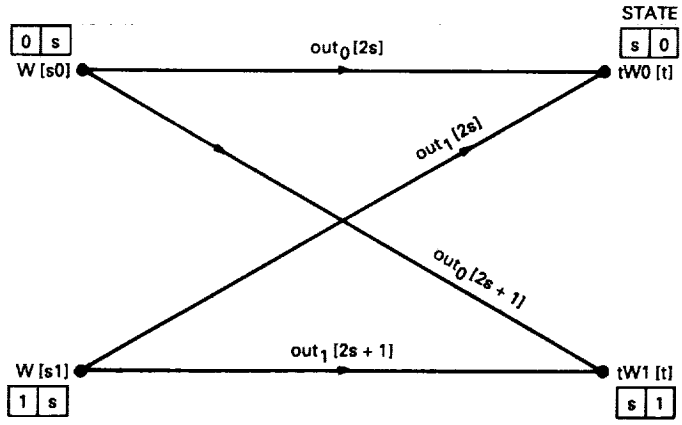

Fig. 4. A rate 1/*n* code butterfly (number *s*).

Appendix

Generating Functions for the (7,1/2) NASA Code

Using the determinant algorithm in section IV, the complete path enumerator $T(D,L,I)$ for the $m=6$, rate 1/2 NASA code was found to contain 1529 numerator and 2799 denominator trivariate terms. The code's weight enumerator, $T(D)=T(D,1,1)$, is

$$11D^{10} -6D^{12} -25D^{14} +D^{16} +93D^{18} -15D^{20} -176D^{22} -76D^{24} +243D^{26} +417D^{28} -228D^{30} -1156D^{32} -49D^{34} +2795D^{36} +611D^{38} -5841D^{40} -1094D^{42} +9575D^{44} +1097D^{46} -11900D^{48} -678D^{50} +11218D^{52} +235D^{54} -8068D^{56} -18D^{58} +4429D^{60} -20D^{62} -1838D^{64} +8D^{66} +562D^{68} -D^{70} -120D^{72} +16D^{76} -D^{80}$$

$$1 - 4D^2 -6D^4 -30D^6 +40D^8 +85D^{10} -81D^{12} -345D^{14} +262D^{16} +844D^{18} -403D^{20} -1601D^{22} +267D^{24} +2509D^{26} +389D^{28} -3064D^{30} -2751D^{32} +2807D^{34} +8344D^{36} -1960D^{38} -16133D^{40} +1184D^{42} +21746D^{44} -782D^{46} -21403D^{48} +561D^{50} +15763D^{52} -331D^{54} -8766D^{56} +131D^{58} +3662D^{60} -30D^{62} -1123D^{64} +3D^{66} +240D^{68} -32D^{72} +2D^{76}$$

$$= 11D^{10} + 38D^{12} + 193D^{14} + 1331D^{16} + 7275D^{18} + 40406D^{20} + \cdots$$

The other two generating functions used to compute error bounds, $\partial T(D,L,I)/\partial L$ and $\partial T(D,L,I)/\partial I$ at $L=I=1$, both have denominators equal to the square of $T(D)$'s denominator above. Their numerators are, respectively,

$$121D^{10} -387D^{12} -706D^{14} +1460D^{16} +3970D^{18} -6157D^{20} -11643D^{22} +8725D^{24} +28677D^{26} +12195D^{28} +88D^{30} -170654D^{32} -306124D^{34} +817895D^{36} +1637616D^{38} -2879440D^{40} -6106837D^{42} +8568521D^{44} +18636083D^{46} -22431469D^{48} -48921504D^{50} +52678351D^{52} +113105887D^{54} -112260733D^{56} -232580537D^{58} +217337170D^{60} +426400859D^{62} -379787502D^{64} -696667758D^{66} +592954735D^{68} +1013294336D^{70} -815739185D^{72} -1311124721D^{74} +968225450D^{76} +1509511967D^{78} -955561827D^{80} -1548537967D^{82} +721812022D^{84} +1419185285D^{86} -302418615D^{88} -1166004400D^{90} -173248817D^{92} +861869027D^{94} +545751792D^{96} -574515412D^{98} -713158178D^{100} +345328990D^{102} +676371119D^{104} -186475599D^{106} -515274530D^{108} +89761092D^{110} +326707300D^{112} -38067910D^{114} -174942675D^{116} +14010022D^{118} +79516060D^{120} -4391424D^{122} -30654965D^{124} +1145504D^{126} +9969013D^{128} -241287D^{130} -2706667D^{132} +39344D^{134} +603670D^{136} -4651D^{138} -107908D^{140} +354D^{142} +14883D^{144} -13D^{146} -1488D^{148} +96D^{152} -3D^{156}$$

and

$$36D^{10} -77D^{12} -140D^{14} +813D^{16} +269D^{18} -4414D^{20} +321D^{22} +14884D^{24} -5273D^{26} -40509D^{28} +39344D^{30} +83884D^{32} -177469D^{34} -111029D^{36} +608702D^{38} -29527D^{40} -1820723D^{42} +817086D^{44} +4951082D^{46} -3436675D^{48} -12279246D^{50} +10300306D^{52} +27735007D^{54} - 25648025D^{56} -56773811D^{58} +55659125D^{60} +104376199D^{62} -106695512D^{64} -170819460D^{66} +180836818D^{68} +247565043D^{70} -270555690D^{72} -317381295D^{74} +356994415D^{76} +360595622D^{78} -415401723D^{80} -364292177D^{82} +426295756D^{84} +328382391D^{86} -385686727D^{88} -264812337D^{90} +307287819D^{92} +191225378D^{94} -215144035D^{96} -123515898D^{98} +131946573D^{100} +71124860D^{102} -70570661D^{104} -36310569D^{106} +32722089D^{108} +16308558D^{110} -13052172D^{112} -6380604D^{114} +4433332D^{116} +2147565D^{118} -1265046D^{120} -612040D^{122} +297721D^{124} +144665D^{126} -56305D^{128} -27569D^{130} +8232D^{132} +4066D^{134} -874D^{136} -435D^{138} +60D^{140} +30D^{142} -2D^{144} -D^{146}$$