The Trellis Complexity of Convolutional Codes

R. J. McEliece
Communications Systems and Research Section
and
California Institute of Technology
Pasadena, California

W. Lin[1]

It has long been known that convolutional codes have a natural, regular trellis structure that facilitates the implementation of Viterbi's algorithm [30,10]. It has gradually become apparent that linear block codes also have a natural, though not in general a regular, "minimal" trellis structure, which allows them to be decoded with a Viterbi-like algorithm [2,31,22,11,27,14,12,16,24,25,8,15]. In both cases, the complexity of the Viterbi decoding algorithm can be accurately estimated by the number of trellis edges per encoded bit. It would, therefore, appear that we are in a good position to make a fair comparison of the Viterbi decoding complexity of block and convolutional codes. Unfortunately, however, this comparison is somewhat muddled by the fact that some convolutional codes, the punctured convolutional codes [4], are known to have trellis representations that are significantly less complex than the conventional trellis. In other words, the conventional trellis representation for a convolutional code may not be the minimal trellis representation. Thus, ironically, at present we seem to know more about the minimal trellis representation for block than for convolutional codes. In this article, we provide a remedy, by developing a theory of minimal trellises for convolutional codes. (A similar theory has recently been given by Sidorenko and Zyablov [29].) This allows us to make a direct performance-complexity comparison for block and convolutional codes. A by-product of our work is an algorithm for choosing, from among all generator matrices for a given convolutional code, what we call a trellis-minimal generator matrix, from which the minimal trellis for the code can be directly constructed. Another by-product is that, in the new theory, punctured convolutional codes no longer appear as a special class, but simply as high-rate convolutional codes whose trellis complexity is unexpectedly small.

I. Introduction

We begin with the standard definition of a convolutional code [9,26], always assuming that the underlying field is $F = GF(2)$. An (n, k) convolutional code $\mathcal{C}$ is a k-dimensional subspace of $F(D)^n$, where $F(D)$ is the field of rational functions in the indeterminate D over the field F. The memory, or degree,

[1] Graduate student at the California Institute of Technology, Pasadena, California.

of $\mathcal{C}$, is the smallest integer m such that $\mathcal{C}$ has an encoder requiring only m delay units. An (n, k) convolutional code with memory m is said to be a (n, k, m) convolutional code. The free distance of $\mathcal{C}$ is the minimum Hamming weight of any codeword in $\mathcal{C}$. An (n, k, m) convolutional code with free distance d is said to be an (n, k, m, d) code.

A minimal generator matrix $G(D)$ for an (n, k, m) convolutional code $\mathcal{C}$ is a $k \times n$ matrix with polynomial entries, whose row space is $\mathcal{C}$, such that the direct-form realization of an encoder for $\mathcal{C}$ based on $G(D)$ uses exactly m delay elements [9,26]. From a minimal generator matrix $G(D)$, or rather from a physical encoder built using $G(D)$ as a blueprint, it is possible to construct a conventional trellis representation for $\mathcal{C}$. This trellis is, in principle, infinite, but it has a very regular structure, consisting (after a short initial transient) of repeated copies of what we shall call the "trellis module" associated with $G(D)$. The trellis module consists of 2^m initial states and 2^m final states, with each initial state being connected by a directed edge to exactly 2^k final states. Thus, the trellis module has 2^{k+m} edges. Each edge is labeled with an n-bit binary vector, namely, the output produced by the encoder in response to the given state transition. Thus, each edge has length (measured in edge labels) n, and so the total edge length of the conventional trellis module is $n2^{k+m}$. Since each trellis module represents the encoder's response to k input bits, we are led to define the conventional trellis complexity of the trellis module as

$$\frac{n}{k} \cdot 2^{m+k} \quad \text{edge labels per encoded bit} \tag{1}$$

or edges per bit, for short. If the code $\mathcal{C}$ is decoded using Viterbi's maximum-likelihood algorithm on the trellis [30,10], the work factor involved in updating the metrics and survivors at each trellis module is proportional to the edge length of the trellis module, so that the trellis complexity as defined in Eq. (1) is a measure of the effort *per decoded bit* required by Viterbi's algorithm. (For a more detailed discussion of the complexity of Viterbi's algorithm on a trellis, see [25, Section 2].)

For example, consider the $(3, 2, 2)$ convolutional code with minimal generator matrix given by

$$G_1(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{pmatrix} \tag{2}$$

This code has the largest possible free distance, viz., $d_{\text{free}} = 3$, for any $(3, 2, 2)$ code. A "direct-form" encoder based on the generator matrix $G_1(D)$ is shown in Fig. 1. If the input pair is (u_1, u_2) and the state of the encoder is (s, t), then the output (x_1, x_2, x_3) is given by

$$\left.\begin{aligned} x_1 &= u_1 + s + t \\ x_2 &= u_1 + s \\ x_3 &= u_1 + u_2 + t \end{aligned}\right\} \tag{3}$$

and the "next state" is just the input pair (u_1, u_2). The conventional trellis module for the code with minimal generator matrix $G_1(D)$ given in Eq. (2) is shown in Fig. 2. The three-bit edge label on the edge from (s, t) to (u_1, u_2) is the triple (x_1, x_2, x_3) given in Eq. (3). The total edge length is 48, so that the conventional trellis complexity corresponding to the matrix $G_1(D)$ is $48/2 = 24$ edges per bit, as predicted by Eq. (1).

But we can do substantially better than this, if we use the fact that this particular code is a punctured convolutional code. We now briefly review the theory of punctured convolutional codes to see how simplified trellises result.

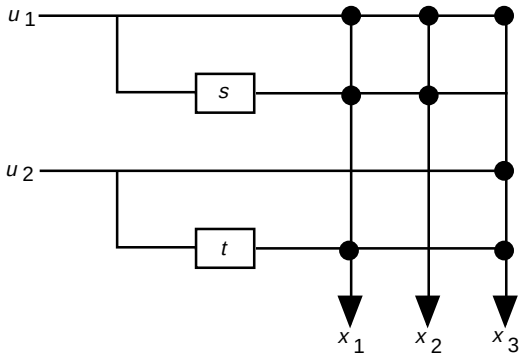

Fig. 1. A direct-form encoder based on the generator matrix $G_1(D)$ in Eq. (2). The input is (u_1, u_2), the output is (x_1, x_2, x_3), and the state of the encoder is (s, t). (The boxes labeled s and t are unit delay elements.)

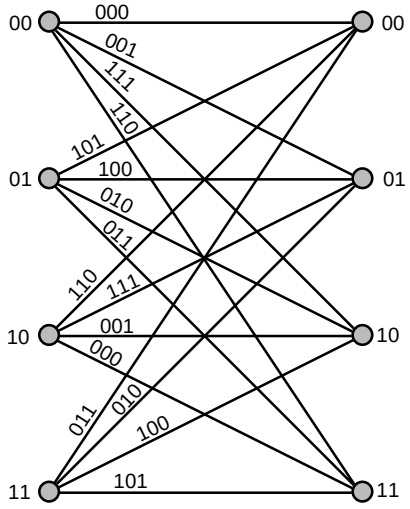

Fig. 2. The conventional trellis module for the code with minimal generator matrix $G_1(D)$ given in Eq. (2).

If we begin with a parent $(N, 1, m)$ convolutional code, and block it to depth k, i.e., group the input bit stream into blocks of k bits each, the result is an (Nk, k, m) convolutional code. If we now delete, or puncture, all but n bits from each Nk-bit output block, the result is an (n, k, m) convolutional code.[2] This punctured code can be represented by a trellis whose trellis module is built from k copies of the trellis modules from the parent $(N, 1, m)$ code, each of which has only 2^{m+1} edges, so that the total number of edge labels on the trellis module is $n \cdot 2^{m+1}$, which means that the trellis complexity of an (n, k, m) punctured code is

$$\frac{n}{k} \cdot 2^{m+1} \quad \text{edges per bit} \tag{4}$$

which is a factor of 2^{k-1} smaller than the complexity of the conventional trellis given in Eq. (1). For $k = 1$, this is no improvement, but for larger values of k, the decoding complexity reduction afforded

[2] In fact, the memory of the punctured code may be less than m, but for most interesting punctured codes, no memory reduction will take place.

by puncturing becomes increasingly significant. And while the class of punctured convolutional codes is considerably smaller than the class of unrestricted convolutional codes, nevertheless many punctured convolutional codes with good performance properties are known [4,13,3,7], and punctured convolutional codes, especially high-rate ones, are often preferred in practice.

For example, consider the $(2, 1, 2, 5)$ convolutional code defined by the minimal generator matrix

$$G_2(D) = (\, 1 + D + D^2 \quad 1 + D^2 \,) \tag{5}$$

The conventional trellis module for this code is shown in Fig. 3. If we block this code into blocks of size $k = 2$, we obtain a $(4, 2, 2)$ convolutional code, still with $d_{\text{free}} = 5$, for which the conventional trellis module is two copies of the trellis module shown in Fig. 3; see Fig. 4.

Now we can do the puncturing. Take the $(4, 2, 2)$ code, as represented by the trellis module in Fig. 4, and delete the second output bit on each of the edges in the second part of the module. The result is shown in Fig. 5. This structure can be thought of as the trellis module for a $(3, 2, 2)$ code; the corresponding d_{free} turns out to be 3. According to Eq. (1), the conventional trellis complexity of a $(3, 2, 2)$ code is $3/2 \cdot 2^4 = 24$ edges per bit. But if we use instead the punctured trellis corresponding to the $k = 2$ blocked version of the parent $(2, 1, 2)$ code, we find from Eq. (4), or Fig. 5, that the trellis complexity is instead only $3/2 \cdot 2^3 = 12$ edges per bit. In fact, it can be shown that this punctured $(3, 2, 2)$ code is the same as the conventional code with generator matrix $G_1(D)$ given in Eq. (2). (Indeed, this example is taken almost verbatim from [4], where it was used to illustrate the way puncturing can reduce decoding complexity.)

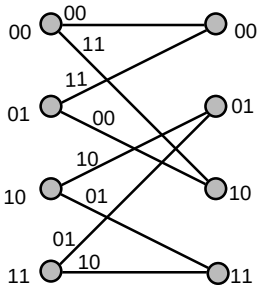

Fig. 3. The trellis module for the (2,1,2) code with generator matrix $G_2(D) = (1 + D + D^2 \quad 1 + D^2)$; total edge length is 16, so the trellis complexity is 16 edges per bit.

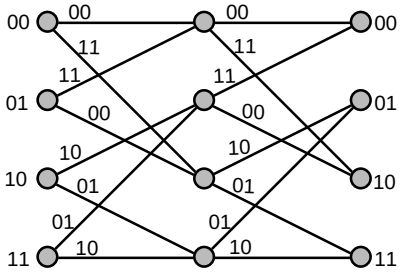

Fig. 4. The trellis module for the (4,2,2) code obtained from the code of Fig. 3 by blocking the inputs in blocks of size 2; total edge length is 32, so the trellis complexity is 32/2 = 16 edges per bit, the same as for the original code.

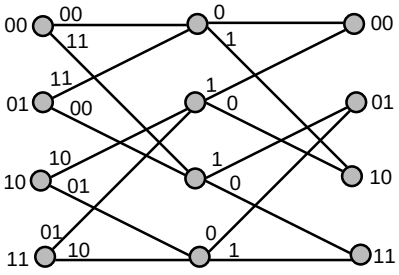

Fig. 5. The trellis module for the (3,2,2) punctured code obtained from the code of Fig. 4 by deleting every fourth bit; total edge length is 16 + 8 = 24, so the trellis complexity is 24/2 = 12 edges per bit.

It seems mysterious that an ordinary-looking generator matrix like $G_1(D)$ produces a code whose trellis complexity can be significantly reduced (if one knows that it is, in fact, a punctured code), whereas for an almost identical code, say one defined by the generator matrix

$$\begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{pmatrix}$$

no such reduction is apparently possible. In Section II, we will resolve this mystery by developing a simple algorithm for constructing the minimum possible trellis for any convolutional code. Our technique will always find a simplified trellis for a punctured code, with complexity at least as small as that given by Eq. (4), even if we are not told in advance that the code can be obtained by puncturing. But more

important, it will often result in considerable simplification of the trellis representation of a convolutional code that is not a punctured code. We will illustrate this with worked examples in Sections II and III and numerical tables in the Appendix.

II. Construction of Minimal Trellises

If $G(D)$ is a minimal generator matrix for an (n, k, m) convolutional code $\mathcal{C}$, then we can write $G(D)$ in the form

$$G(D) = G_0 + G_1 D + \cdots + G_L D^L \tag{6}$$

where $G_0, \cdots, G_L$ are $k \times n$ scalar matrices (i.e., matrices whose entries are from $GF(2)$), and L is the maximum degree of any entry of $G(D)$. If we concatenate the $L+1$ matrices $G_0, \cdots, G_L$, we obtain a $k \times (L+1)n$ scalar matrix, which we denote by $\widetilde{G}$:

$$\widetilde{G} = (\, G_0 \quad G_1 \quad \cdots \quad G_L \,) \tag{7}$$

It is well known [23, Chapter 9] that the matrix $\widetilde{G}$ and its shifts can be used to build a scalar generator matrix G_{scalar} for the code $\mathcal{C}$ (for simplicity of notation, we illustrate the case $L = 2$):

$$G_{\text{scalar}} = \begin{bmatrix} G_0 & G_1 & G_2 & & & \\ & G_0 & G_1 & G_2 & & \\ & & G_0 & G_1 & G_2 & \\ & & & G_0 & G_1 & G_2 \\ & & & & \ddots & \end{bmatrix} \tag{8}$$

The matrix in Eq. (8) is, except for the fact that it continues forever, the generator matrix for a binary block code (with a very regular structure), and so the techniques that have been developed for finding minimal trellises for block codes are useful for constructing trellis representations for convolutional codes. Here we apply the techniques developed in [25, Section 7], which show how to construct a trellis directly from any generator matrix for a given block code, and the minimal trellis if the generator is in minimal span form, to construct a trellis for $\mathcal{C}$ based on the infinite scalar generator matrix G_{scalar}.

The trellis module for the trellis associated with G_{scalar} corresponds to the $(L+1)k \times n$ matrix module,

$$\widehat{G} = \begin{pmatrix} G_L \\ G_{L-1} \\ \vdots \\ G_0 \end{pmatrix} \tag{9}$$

which repeatedly appears as a vertical "slice" in G_{scalar}. Using the techniques in [25, Section 7], it is easy to show that the number of edges in this trellis module is

$$\text{edge count} = \sum_{j=1}^{n} 2^{a_j} \tag{10}$$

where a_j is the number of active entries in the jth column of the matrix module $\widehat{G}$. (An element is called active if it belongs to the active span of one of the rows of $\widetilde{G}$. We will elaborate on this below.) Our object, then, is to find a generator matrix for which the edge count in the corresponding trellis module is as small as possible.

To clarify these ideas, we consider the $(3, 2, 1)$ code with (minimal) generator matrix

$$G_3(D) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1+D & 1+D \end{pmatrix} \tag{11}$$

According to Eq. (1), the conventional trellis complexity for this code is 12 edges per bit. However, we can do better. The scalar matrix $\widetilde{G_3}$ corresponding to $G_3(D)$ is [cf. Eq. (7)]

$$\widetilde{G_3} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix} \tag{12}$$

In Eq. (12), we have shown the active elements of each row, i.e., the entries from the first nonzero entry to the last nonzero entry, in boldface. The span length of (i.e., the number of active entries in) the first row is, therefore, three; and the span length of the second row is six. The matrix module corresponding to $\widetilde{G_3}$ is [cf. Eq. (9)]

$$\widehat{G_3} = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

Thus, $a_1 = 3$, $a_2 = 3$, and $a_3 = 3$, which by Eq. (10) means that the corresponding trellis module has $2^3 + 2^3 + 2^3 = 24$ edges. Since each trellis module represents two encoded bits, the resulting trellis complexity is $24/2 = 12$ edges per bit. Since we have already noted that the conventional trellis complexity for this code is also 12 edges per bit, the trellis corresponding to $G_3(D)$ is not better than (in fact, it is isomorphic to) the conventional trellis. To do better, we need to find a generator matrix for the code for which $\sum_i 2^{a_i}$ is less than 24. Using the results of [25, Section 6], it is possible to show that minimizing $\sum_i 2^{a_i}$ is equivalent to minimizing $\sum_i a_i$, i.e., the total span length of the corresponding $\widetilde{G}$, and so we shall look for generator matrices for which the span of $\widetilde{G}$ is reduced.

Note that if we add the first row of $G_3(D)$ to the second row, the resulting generator matrix, which is still minimal, is

$$G'_3(D) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1+D & D \end{pmatrix}$$

The scalar matrix $\widetilde{G'_3}$ corresponding to $G'_3(D)$ is [cf. Eq. (7)]

$$\widetilde{G'_3} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix} \tag{13}$$

The span length of the first row of $G'_3(D)$ is three, and the span length of the second row is five, and so the total span length is eight, one less than that of $G_3(D)$. The matrix module corresponding to $\widetilde{G'_3}$ is [cf. Eq. (9)]

$$\widehat{G'_3} = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

Here $a_1 = 2$, $a_2 = 3$, and $a_3 = 3$, and so by Eq. (10) the corresponding trellis module has $2^2+2^3+2^3 = 20$ edges, so that the resulting trellis complexity is $20/2 = 10$ edges per bit. The trellis module itself, constructed using the technique described in [25, Section 7] is shown in Fig. 6.

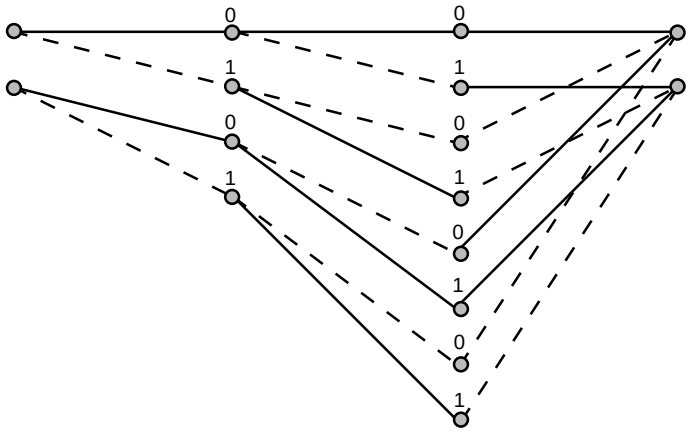

Fig. 6. The trellis module for the (3,2,1) code with generator matrix $G'_3(D)$. (Solid edges represent "0" code bits, and dashed edges represent "1" code bits. The labels on the vertices correspond to the information bits.)

But we can do still better. If we multiply the first row of $G'_3(D)$ by D and add it to the second row, the resulting generator matrix, which is still minimal, is

$$G''_3(D) = \begin{pmatrix} 1 & 0 & 1 \\ D & 1+D & 0 \end{pmatrix}$$

The scalar matrix $\widetilde{G''_3}$ corresponding to $G''_3(D)$ is [cf. Eq. (7)]

$$\widetilde{G''_3} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 \end{pmatrix} \tag{14}$$

The span length of $G''_3(D)$ is seven, one less than that of $G'_3(D)$. The matrix module corresponding to $\widetilde{G''_3}$ is [cf. Eq. (9)]

$$\widehat{G''_3} = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

Here $a_1 = 2$, $a_2 = 3$, and $a_3 = 2$, and so by Eq. (10), the corresponding trellis module has $2^2+2^3+2^2 = 16$ edges, so that the resulting trellis complexity is $16/2 = 8$ edges per bit. The trellis module itself, again constructed using the techniques described in [25, Section 7] is shown in Fig. 7.

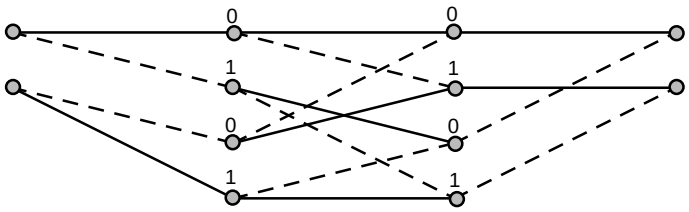

Fig. 7. The trellis module for the (3,2,1) code with generator matrix $G_3^{\prime\prime\prime}(D)$. This is the minimal trellis module for this code.

Furthermore, it is easy to see that there is no generator matrix for this code with span length less than seven, so that the trellis module shown in Fig. 7 yields the minimal trellis for the code. Alternatively, we examine the scalar generator matrix for the code corresponding to $\widetilde{G_3''}$ [cf. Eq. (8)]:

$$G_{\text{scalar}} = \begin{bmatrix} \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & & & & & \\ 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & & & & & \\ & & & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & & \\ & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & & \\ & & & & & & 0 & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & \\ & & & & & & & & & \ddots & & & \end{bmatrix} \tag{15}$$

In Eq. (15), we see that G_{scalar} has the property that no column contains more than one underlined entry, the leftmost nonzero entry in its row (L), or more than one overlined entry, the rightmost nonzero entry in its row (R). Thus, G_{scalar} has the LR property, and so, *if it were a finite matrix*, it would produce the minimal trellis for the code [25, Sec. 6]. To circumvent the problem that G_{scalar} is infinite, we can define the Mth truncation of the code $\mathcal{C}$, denoted by $\mathcal{C}^{[M]}$, as the $((M+L)n, Mk)$ block code obtained by taking only the first Mk rows of G_{scalar}, i.e., the code with $Mk \times (M+L)n$ generator matrix

$$G_{\text{scalar}}^{[M]} = \begin{bmatrix} \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & & & & \\ 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & & & & \\ & & & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & \\ & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & \\ & & & & & & & & & \ddots & & \\ & & & & & & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 \end{bmatrix} \tag{16}$$

Plainly, if G_{scalar} has the LR property, so does $G_{\text{scalar}}^{[M]}$, for all $M \geq 1$. Thus, it follows from the standard theory of trellises for block codes that the matrix $G_{\text{scalar}}^{[M]}$ produces the minimal trellis for $\mathcal{C}^{[M]}$, for all M, and so we can safely call the infinite trellis, built from trellis modules corresponding to $\widehat{G}$, *the* minimal trellis for the code. (Note that, in this example, the ratio of the conventional trellis complexity to the minimal trellis complexity is $12/8 = 3/2$. If this code were punctured, then according to Eqs. (1) and (4), the ratio would be at least 2. Thus, we conclude that the code with generator matrix $G_3(D)$ as given in Eq. (11) is not a punctured code, which shows that the theory of minimal trellises for convolutional codes goes beyond merely "explaining" punctured codes.)

The preceding argument, though it was presented in terms of a specific example, is entirely general. It shows that a basic generator matrix $G(D)$ produces a minimal trellis if and only if $G(D)$ has the property that the span length of the corresponding $\widetilde{G}$ cannot be reduced by an operation of the form

$$g_i(D) \leftarrow g_i(D) + D^\ell g_j(D)$$

where $g_i(D)$ is the ith row of $G(D)$ and ℓ is an integer in the range $0 \le \ell \le L$. We shall call a generator matrix with this property a trellis-minimal generator matrix for $\mathcal{C}$. A trellis-minimal generator matrix must be minimal, but the converse need not be true, as the example of this section shows. Furthermore, it can be shown that the set of trellis-minimal generator matrices for a given code $\mathcal{C}$ coincides with the set of generator matrices for which the span length of the corresponding $\widetilde{G}$ is a minimum. In the next section, we will give two more examples of minimal trellises.

III. Two More Examples

Our first example is for the code whose generator matrix is given in Eq. (2). The corresponding decomposition [cf. Eq. (6)] is

$$G_1(D) = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} D$$

The scalar matrix $\widetilde{G}$ is, thus,

$$\widetilde{G}_1 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

and the matrix module $\widehat{G}$ from Eq. (9) is then

$$\widehat{G}_1 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} \end{pmatrix} \tag{17}$$

Since there are three active entries in each column of $\widehat{G}$, it follows from Eq. (10) that the edge count for the trellis module is $2^3 + 2^3 + 2^3 = 24$, so that the trellis complexity for this trellis module is $24/2 = 12$ edges per bit, the same as given by Eq. (4) for the punctured trellis. To actually construct the trellis module, we can use the techniques of [25, Section 7], and the result is shown in Fig. 8. Finally, we note that the G_{scalar} corresponding to the matrix $\widehat{G}_1$ of Eq. (17) is [cf. Eq. (8)]

$$\begin{bmatrix}
\underline{1} & 1 & 1 & 1 & \overline{1} & 0 & & & & & & \\
0 & 0 & \underline{1} & 1 & 0 & \overline{1} & & & & & & \\
 & & & \underline{1} & 1 & 1 & 1 & \overline{1} & 0 & & & \\
 & & & 0 & 0 & \underline{1} & 1 & 0 & \overline{1} & & & \\
 & & & & & & \underline{1} & 1 & 1 & 1 & \overline{1} & 0 \\
 & & & & & & 0 & 0 & \underline{1} & 1 & 0 & \overline{1} \\
 & & & & & & & \ddots & & & &
\end{bmatrix}$$

which has the LR property, and so $G_1(D)$ is trellis-minimal. (This code is the first code listed in Table 2 in the Appendix.)

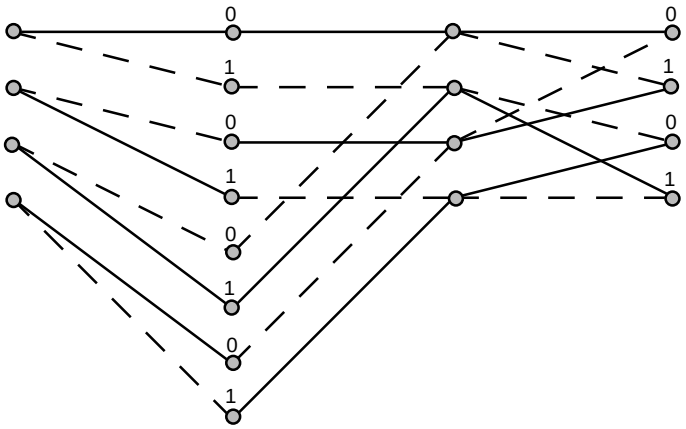

Fig. 8. The trellis module for the (3,2,2) code with generator matrix $G_1(D)$. This module is isomorphic to the one in Fig. 5.

As our second example, we consider a partial-unit-memory code, taken from [20,1]. It is an $(8, 4, 3)$ code with $d_{\text{free}} = 8$ and with minimal generator matrix (as taken from [1])

$$G(D) = \begin{pmatrix} 11111111 \\ 11101000 \\ 10110100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10101100 \\ 10010110 \end{pmatrix} D \tag{18}$$

The conventional trellis complexity for this code is, by Eq. (1), $8/4 \cdot 2^7 = 256$ edges per bit. We can reduce this number to 120, as follows. First, we concatenate the two matrices in Eq. (18), obtaining the following 4×16 scalar matrix $\widetilde{G}$:

$$\widetilde{G} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Next, using the techniques developed in [25, Section 6], we perform a series of elementary row operations on $\widetilde{G}$, transforming it to the minimal span, or trellis oriented form, $\widetilde{G}'$:

$$\widetilde{G}' = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 \end{pmatrix} \tag{19}$$

The matrix module $\widehat{G}$ defined in Eq. (9) is, thus,

$$\widehat{G} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

and so by Eq. (10) the total edge length of the trellis module is $2^4 + 2^5 + 2^6 + 2^7 + 2^7 + 2^6 + 2^5 + 2^4 = 480$. Since each trellis module represents four encoded bits, it follows that the trellis complexity is $480/4 = 120$ edges per bit, compared to the conventional trellis complexity, cited above, of 256 edges per bit.

The matrix G_{scalar} corresponding to the matrix $\widetilde{G}'$ in Eq. (19) is easily seen to have the LR property, and so the generator matrix [cf. Eq. (19)]

$$G'(D) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} D$$

is trellis-minimal. However, the trellis complexity can be reduced still further, if we allow column permutations of the original generator matrix $G(D)$ in Eq. (18). Indeed, by computer search, we have found that one minimal complexity column permutation for this particular code is the permutation (01243567), which results in the generator matrix [cf. Eq. (18)]

$$G(D) = \begin{pmatrix} 11111111 \\ 11110000 \\ 10101100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10110100 \\ 10001110 \end{pmatrix} D \tag{20}$$

Then, after putting the minimal generator matrix of Eq. (20) into trellis-minimal form, it becomes

$$G(D) = \begin{pmatrix} 11111111 \\ 00001111 \\ 01111111 \\ 00111111 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11111000 \\ 11111100 \\ 11111110 \end{pmatrix} D \tag{21}$$

The trellis complexity of the generator matrix in Eq. (21) turns out to be 104 edges per encoded bit. (This code is the seventh code listed in Table 6 in the Appendix.) The minimal trellis complexity of unit memory and partial unit memory convolutional codes has also been studied in [6] and [32].

IV. LTC Versus ACG

In this section, we will attempt to compare the trellis complexity of a number of codes to their performance. To do this, we define the logarithmic trellis complexity (LTC) of a code, block or convolutional, as the base-2 logarithm of the minimal trellis complexity (edges per encoded bit) and the asymptotic coding gain (ACG) as the code's rate times its minimum (or free) distance. An empirical study, based on existing tables of convolutional codes (e.g., the tables in [19,28,20,5,7]), reveals the interesting fact that LTC / ACG lies between 1.5 and 2.0 for most "good" convolutional codes. For example, for the $(3, 2, 2, 3)$

code discussed in Section III, the ratio is 1.79, and for the $(8, 4, 3, 8)$ code, it is 1.68. By comparison, for the "NASA standard" $(2, 1, 6, 10)$ convolutional code, for which, as for all $(n, 1, m)$ convolutional codes, the minimal trellis complexity is given by the formula of Eq. (1), the ratio is 1.60. In the Appendix, we list the (ACG, LTC) pairs for a large number of convolutional codes and a few block codes. In Fig. 9, we show a scatter plot of these pairs. It is interesting to note how close most of these pairs are to the line of slope 2. This experimental fact may be related to a recent theorem of Lafourcade and Vardy [18], which implies that for any sequence of block codes with a fixed rate $R > 0$ and fixed value of $d/n > 0$, as $n \to \infty$,

$$\liminf_{n\to\infty} \frac{\text{LTC}}{\text{ACG}} \geq 2 \tag{22}$$

In any case, we have been able to show that for all codes, the ratio LTC / ACG must be strictly greater than 1. (This result is similar to Theorem 3 in [17].)

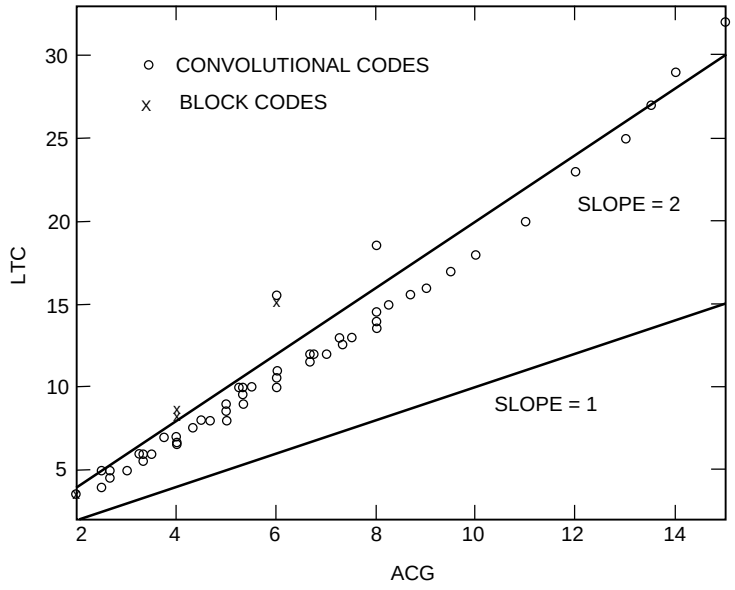

Fig. 9. A scatter plot of the pairs (ACG, LTC) for the codes listed in the Appendix.

V. Conclusion and Open Problems

In this article, we have shown that every convolutional code has a unique minimal trellis representation, which is in many cases considerably simpler than the conventional trellis for the code. We have also presented a simple technique for actually constructing the minimal trellis for any convolutional code, and we have numerically computed the trellis complexity for many convolutional codes. In principle, the theory of minimal trellises for convolutional codes can be deduced from the general Forney–Trott theory [12], but we believe the observation that the Viterbi decoding complexity of many convolutional codes, including many nonpunctured codes, can thereby be reduced systematically is new, as are the details of the algorithms for producing the minimal trellises.

We close with a list of research problems that suggest themselves.

(1) A given convolutional code will, in general, have many different minimal generator matrices [21], but as we saw in Section II, not all minimal generator matrices are trellis minimal. What can be said about the class of trellis minimal generator matrices?

(2) A theoretical explanation of the experimental observation that most of the codes shown in Fig. 9 lie near the line of slope 2 would be welcome.

(3) The design and implementation of Viterbi's algorithm on conventional trellises is well understood. Since the techniques described here lead to greatly reduced trellis complexity, it would be worthwhile to make a careful study of how best to implement Viterbi's algorithm on minimal trellises.

(4) From our current viewpoint, punctured convolutional codes are just codes whose trellis module has fewer edges than would normally be expected. Indeed, it is easy to prove that the minimal trellis complexity of any punctured convolutional code is at least as small as the punctured trellis complexity given in Eq. (4). This is because in the scalar matrix $\widetilde{G}$ for a punctured code, certain entries are guaranteed to be zero. For example, for a $(4, 3, 3)$ punctured code, the matrix $\widetilde{G}$ has the template structure

$$\widetilde{G} = \begin{pmatrix} x & x & x & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & x & x & 0 \\ 0 & 0 & 0 & x & x & x & x & x \end{pmatrix}$$

where the x's can be arbitrary (actually, there are restrictions on the x's that depend in detail on how the code is constructed), but the eight zero positions must be respected. Any $(4, 3, 3)$ convolutional code with such a template structure will have trellis complexity at most $4/3 \cdot 2^4 = 211/3$. An obvious question is whether other low complexity templates support good convolutional codes.

(5) In our computer-aided search for the "best" column permutation of the $(8, 4, 3, 8)$ code, we found that each of the $8! = 40,326$ possible column permutations had minimal trellis complexity of either 120 or 104. This strongly suggests an equivalence among permutations that, if understood theoretically, could make it much simpler to find the best column permutation.

Finally, we remark that when the bulk of this article was written, we were not aware of the important earlier work of Sidorenko and Zyablov [29], which deals explicitly with the minimal trellis for a convolutional code, and we wish to acknowledge their priority. Their work, like ours, develops the theory of minimal trellises for convolutional codes from the corresponding theory for block codes. However, their trellis construction is based on the parity-check matrix of the code rather than the generator matrix, and their emphasis is quite different. One advantage of the Sidorenko–Zyablov approach is that it leads to the following upper bound on the number of nodes at depth i in the minimal trellis for a (n, k, m) convolutional code [29, Theorem 1]:

$$N_i \leq 2^{m+\min(k,n-k)}$$

It is not easy to derive this bound using our methods. On the other hand, the present article contains a number of things not present in [29], among them being

(1) The observation that the minimal trellis for a punctured convolutional code is at least as simple as the punctured trellis.

(2) The concept of a trellis-minimal generator matrix for a convolutional code, and an algorithm for computing one.

(3) The ACG versus LTC comparison for block and convolutional codes.

References

[1] K. Abdel-Ghaffar, R. J. McEliece, and G. Solomon, "Some Partial Unit Memory Convolutional Codes," *The Telecommunications and Data Acquisition Progress Report 42-107, July–September 1991*, Jet Propulsion Laboratory, Pasadena, California, pp. 57–72, November 15, 1991. Also see *Proc. 1991 International Symposium on Information Theory*, Budapest, p. 196, June 1991.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, March 1974.

[3] G. Bégin and D. Haccoun, "High-Rate Punctured Convolutional Codes: Structure Properties and Construction Technique," *IEEE Trans. Comm.*, vol. COM-37, pp. 1381–1385, December 1989.

[4] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 97–100, January 1979.

[5] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New Short Constraint Length Convolutional Code Construction for Selected Rational Rates," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 794–800, September 1982.

[6] U. Dettmar and U. Sorger, "On Maximum Likelihood Decoding of Unit Memory Codes," *Proc. 6th Swedish–Russian International Workshop on Information Theory*, pp. 184–188, August 1993.

[7] A. Dholakia, *Introduction to Convolutional Codes with Applications*, Boston: Kluwer Academic Publishers, 1994.

[8] S. Dolinar, L. Ekroot, A. Kiely, R. McEliece, and W. Lin, "The Permutation Trellis Complexity of Linear Block Codes," *Proc. 32nd Annual Allerton Conference on Communication, Control, and Computing*, Allerton Park, Illinois, pp. 60–74, September 1994.

[9] G. D. Forney Jr., "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 268–278, November 1970.

[10] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, vol. 61, pp. 268–276, March 1973.

[11] G. D. Forney, Jr., "Coset Codes—Part II: Binary Lattices and Related Codes," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1152–1187, September 1988.

[12] G. D. Forney, Jr., and M. D. Trott, "The Dynamics of Group Codes: State Spaces, Trellis Diagrams, and Canonical Encoders," *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 1491–1513, September 1993.

[13] D. Haccoun and G. Bégin, "High-Rate Punctured Convolutional Codes for Viterbi and Sequential Decoding," *IEEE Trans. Comm.*, vol. COM-37, pp. 1113–1125, November 1989.

[14] B. Honary, G. Markarian, and M. Darnell, "Trellis Decoding for Block Codes," *Proc. 3rd IEE Int. Symp. Comm. Theory Appl.*, Ambleside, United Kingdom, pp. 79–93, July 1993.

[15] A. Kiely, S. Dolinar, R. McEliece, L. Ekroot, and W. Lin, "Trellis Decoding Complexity of Linear Block Codes," to appear in *IEEE Trans. Inform. Theory*, vol. IT-42, November 1996.

[16] F. R. Kschischang and V. Sorokine, "On the Trellis Structure of Block Codes," *IEEE Trans. Inform. Theory*, vol. IT-41, November 1995, in press.

[17] A. Lafourcade and A. Vardy, "Asymptotically Good Codes Have Infinite Trellis Complexity," *IEEE Trans. Inform. Theory*, vol. IT-41, pp. 555–559, March 1995.

[18] A. Lafourcade and A. Vardy, "Lower Bounds on Trellis Complexity of Block Codes,"*IEEE Trans. Inform. Theory*, vol. IT-41, November 1995, in press.

[19] K. Larsen, "Short Convolutional Codes With Maximal Free Distance for Rates 1/2, 1/3, and 1/4," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 371–372, May 1973.

[20] G. S. Lauer, "Some Optimal Partial-Unit-Memory Codes," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 540–547, March 1979.

[21] K. Lumbard and R. J. McEliece, "Counting Minimal Generator Matrices," *Proc. 1994 IEEE Inter. Symp. Inform. Theory*, Trondheim, Norway, p. 18, June 1994.

[22] J. L. Massey, "Foundations and Methods of Channel Coding," *Proc. Int. Conf. Inform. Theory and Systems*, NTG-Fachberichte, vol. 65, pp. 148–157, 1978.

[23] R. J. McEliece, *The Theory of Information and Coding,* Reading, Massachusetts: Addison-Wesley, 1977.

[24] R. J. McEliece, "The Viterbi Decoding Complexity of Linear Block Codes," *Proc. 1994 IEEE Inter. Symp. Inform. Theory*, Trondheim, Norway, p. 341, June 1994.

[25] R. J. McEliece, "On the BCJR Trellis," to appear in *IEEE Trans. Inform. Theory.*, vol. IT-42, 1996.

[26] R. J. McEliece, "The Algebraic Theory of Convolutional Codes," to appear as a chapter in the *Handbook of Coding Theory*, edited by R. A. Brualdi, W. C. Huffman, and V. Pless, Amsterdam: Elsinore Science Publishers, 1996.

[27] D. J. Muder, "Minimal Trellises for Block Codes," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1049–1053, September 1988.

[28] E. Paaske, "Short Binary Convolutional Codes With Maximal Free Distance," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 683–688, September 1974.

[29] V. Sidorenko and V. Zyablov, "Decoding of Convolutional Codes Using a Syndrome Trellis," *IEEE Trans. Inform. Theory*, vol. IT-40, pp. 1663–1666, September 1994.

[30] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, April 1967.

[31] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 76–80, January 1978.

[32] V. Zyablov and V. Sidorenko, "Soft Decision Maximum Likelihood Decoding of Partial Unit Memory Codes," *Problems of Information Transmission*, vol. 28, no. 1, pp. 18–22, July 1992.

Appendix

Tables of LTC Versus ACG

In this appendix, we list the ACG and the LTC for a large number of "good" convolutional codes and a few block codes. A scatter plot of these (ACG, LTC) pairs appears as Fig. 9 in Section IV.

Table 1. Best (2,1,*m*) codes.[a]

Code	LTC	ACG	LTC–ACG ratio
(2,1,2,5)	4	2.5	1.60
(2,1,3,6)	5	3	1.67
(2,1,4,7)	6	3.5	1.71
(2,1,5,8)	7	4	1.75
(2,1,6,10)	8	5	1.60
(2,1,8,12)	10	6	1.67
(2,1,10,14)	12	7	1.71
(2,1,11,15)	13	7.5	1.73
(2,1,12,16)	14	8	1.75
(2,1,14,18)	16	9	1.78
(2,1,15,19)	17	9.5	1.79
(2,1,16,20)	18	10	1.80
(2,1,18,22)	20	11	1.82
(2,1,21,24)	23	12	1.92
(2,1,23,26)	25	13	1.92
(2,1,25,27)	27	13.5	2.00
(2,1,27,28)	29	14	2.07
(2,1,30,30)	32	15	2.13

[a] From pp. 85–88 in [7].

Table 2. Best (3,2,*m*) codes.[a]

Code	LTC	ACG	LTC–ACG ratio
(3,2,2,3)	3.58	2.00	1.79
(3,2,3,4)	5.00	2.67	1.87
(3,2,4,5)	6.00	3.33	1.80
(3,2,5,6)	7.00	4.00	1.75
(3,2,6,7)	8.00	4.67	1.71
(3,2,7,8)	9.00	5.33	1.69
(3,2,8,8)	10.00	5.33	1.88
(3,2,9,9)	11.00	6.00	1.83
(3,2,10,10)	12.00	6.67	1.80

[a] From p. 90 in [7].

Table 3. Best (4,3,*m*) codes.[a]

Code	LTC	ACG	LTC–ACG ratio
(4,3,3,4)	5.00	3.00	1.67
(4,3,5,5)	7.00	3.75	1.87
(4,3,6,6)	8.00	4.50	1.78
(4,3,8,7)	10.00	5.25	1.90
(4,3,9,8)	11.00	6.00	1.83

[a] From p. 90 in [7].

Table 4. Best (3,1,*m*) codes.[a]

Code	LTC	ACG	LTC–ACG ratio
(3,1,2,8)	4.58	2.67	1.72
(3,1,3,10)	5.58	3.33	1.68
(3,1,4,12)	6.58	4.00	1.64
(3,1,5,13)	7.58	4.33	1.75
(3,1,6,15)	8.58	5.00	1.72
(3,1,7,16)	9.58	5.33	1.80
(3,1,8,18)	10.58	6.00	1.76
(3,1,9,20)	11.58	6.67	1.74
(3,1,10,22)	12.58	7.33	1.72
(3,1,11,24)	13.58	8.00	1.70
(3,1,12,24)	14.58	8.00	1.82
(3,1,13,26)	15.58	8.67	1.80

[a] From p. 89 in [7].

Table 5. Best (4,1,*m*) codes.[a]

Code	LTC	ACG	LTC–ACG ratio
(4,1,2,10)	5.00	2.50	2.00
(4,1,3,13)	6.00	3.25	1.85
(4,1,4,16)	7.00	4.00	1.75
(4,1,5,18)	8.00	4.50	1.78
(4,1,6,20)	9.00	5.00	1.80
(4,1,7,22)	10.00	5.50	1.82
(4,1,8,24	11.00	6.00	1.83
(4,1,9,27)	12.00	6.75	1.78
(4,1,10,29)	13.00	7.25	1.79
(4,1,11,32)	14.00	8.00	1.75
(4,1,12,33)	15.00	8.25	1.82
(4,1,13,36)	16.00	9.00	1.78

[a] From p. 89 in [7].

Table 6. Some block codes and partial unit memory convolutional codes.

Code	LTC	ACG	LTC–ACG ratio
[8,4,4] Self-dual code	3.46	2.00	1.73
[24,12,8] Golay code	8.22	4.00	2.06
[32,16,8] BCH[a] code	8.64	4.00	2.16
[48,24,12] Self-dual code	15.13	6.00	2.52
$[n, n-1, 2]$ Parity-check code	2.00	$\frac{2(n-1)}{n}$	$\frac{n}{n-1}$
$[n, 1, n]$ Repetition code	$1 + \log_2 n$	1	$1 + \log_2 n$
(8,4,3,8) PUM[b] code	6.70	4.00	1.68
(24,12,7,12) PUM code	15.58	6.00	2.60
(24,12,10,16) PUM code	18.58	8.00	2.32

[a] Bose–Chaudhuri–Hocquenghem.

[b] Partial unit memory.