Chapter 9

Introduction to convolutional codes

We now introduce binary linear convolutional codes, which like binary linear block codes are useful in the power-limited (low-SNR, low-ρ) regime. In this chapter we will concentrate on rate-$1/n$ binary linear time-invariant convolutional codes, which are the simplest to understand and also the most useful in the power-limited regime. Here is a canonical example:

Example 1. Figure 1 shows a simple rate-1/2 binary linear convolutional encoder. At each time k, one input bit u_k comes in, and two output bits (y_{1k}, y_{2k}) go out. The input bits enter a 2-bit shift register, which has 4 possible states (u_{k-1}, u_{k-2}). The output bits are binary linear combinations of the input bit and the stored bits.

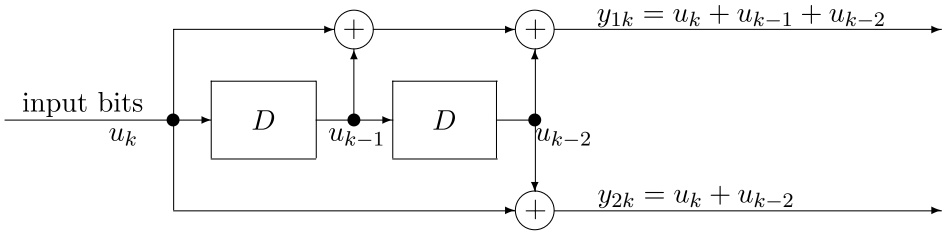

Figure 1. Four-state rate-1/2 binary linear convolutional encoder.

The code $\mathcal{C}$ generated by this encoder is the set of all output sequences that can be produced in response to an input sequence $\mathbf{u} = (\ldots, u_k, u_{k+1}, \ldots)$. The code is linear because if $(\mathbf{y}_1, \mathbf{y}_2)$ and $(\mathbf{y}'_1, \mathbf{y}'_2)$ are the code sequences produced by $\mathbf{u}$ and $\mathbf{u}'$, respectively, then $(\mathbf{y}_1 + \mathbf{y}'_1, \mathbf{y}_2 + \mathbf{y}'_2)$ is the code sequence produced by $\mathbf{u} + \mathbf{u}'$.

As in the case of linear block codes, this implies that the minimum Hamming distance $d_{\rm free}$ between code sequences is the minimum Hamming weight of any nonzero codeword. By inspection, we can see that the Hamming weight of the output sequence when the input sequence is $(\ldots, 0, 0, 1, 0, 0, \ldots)$ is 5, and that this is the only weight-5 sequence starting at a given time k. Of course, by time-invariance, there is such a weight-5 sequence starting at each time k.

We will see that maximum-likelihood sequence decoding of a convolutional code on an AWGN channel can be performed efficiently by the Viterbi algorithm (VA), with complexity proportional to the number of states (in this case 4). As with block codes, the probability of error per bit $P_b(E)$ may be estimated by the union bound estimate (UBE) as

$$P_b(E) \approx K_b(\mathcal{C}) Q^{\surd}\left(\gamma_{\rm c}(\mathcal{C})(2E_b/N_0)\right),$$

where the nominal coding gain is $\gamma_c(\mathcal{C}) = Rd_{\text{free}}$, R is the code rate in input bits per output bit, and $K_b(\mathcal{C})$ is the number of minimum-weight code sequences per input bit. For this code, $d_{\text{free}} = 5$, $R = 1/2$, and $K_b(\mathcal{C}) = 1$, which means that the nominal coding gain is $\gamma_c(\mathcal{C}) = 5/2$ (4 dB), and the effective coding gain is also 4 dB. There is no block code that can achieve an effective coding gain of 4 dB with so little decoding complexity. □

As Example 1 shows, convolutional codes have two different kinds of structure: algebraic structure, which arises from convolutional encoders being linear systems, and dynamical structure, which arises from convolutional encoders being finite-state systems. We will first study their linear system structure. Then we will consider their finite-state structure, which is the key to ML decoding via the VA. Finally, we will show how to estimate performance using the UBE, and will give tables of the complexity and performance of the best known rate-$1/n$ codes.

9.1 Linear time-invariant systems over finite fields

We start with a little linear system theory, namely the theory of linear time-invariant (LTI) systems over finite fields. The reader is probably familiar with the theory of discrete-time LTI systems over the real or the complex field, which are sometimes called discrete-time real or complex filters. The theory of discrete-time LTI systems over an arbitrary field $\mathbb{F}$ is similar, except that over a finite field there is no notion of convergence of an infinite sum.

9.1.1 The input/output map of an LTI system

In general, a discrete-time system is characterized by an input alphabet $\mathcal{U}$, an output alphabet $\mathcal{Y}$, and an input/output map from bi-infinite discrete-time input sequences $\mathbf{u} = (\ldots, u_k, u_{k+1}, \ldots)$ to output sequences $\mathbf{y} = (\ldots, y_k, y_{k+1}, \ldots)$. Here we will take the input and output alphabets to be a common finite field, $\mathcal{U} = \mathcal{Y} = \mathbb{F}_q$. The indices of the input and output sequences range over all integers in $\mathbb{Z}$ and are regarded as time indices, so that for example we may speak of u_k as the value of the input at time k.

Such a system is *linear* if whenever $\mathbf{u}$ maps to $\mathbf{y}$ and $\mathbf{u}'$ maps to $\mathbf{y}'$, then $\mathbf{u} + \mathbf{u}'$ maps to $\mathbf{y} + \mathbf{y}'$ and $\alpha\mathbf{u}$ maps to $\alpha\mathbf{y}$ for any $\alpha \in \mathbb{F}_q$. It is *time-invariant* if whenever $\mathbf{u}$ maps to $\mathbf{y}$, then $D\mathbf{u}$ maps to $D\mathbf{y}$, where D represents the *delay operator*, namely the operator whose effect is to delay every element in a sequence by one time unit; *i.e.,* $\mathbf{u}' = D\mathbf{u}$ means $\mathbf{u}'_k = \mathbf{u}_{k-1}$ for all k.

It is well known that the input/output map of an LTI system is completely characterized by its *impulse response* $\mathbf{g} = (\ldots, 0, 0, \ldots, 0, g_0, g_1, g_2, \ldots)$ to an input sequence $\mathbf{e}_0$ which is equal to 1 at time zero and 0 otherwise. The expression for $\mathbf{g}$ assumes that the LTI system is *causal*, which implies that the impulse response $\mathbf{g}$ must be equal to 0 before time zero.

The proof of this result is as follows. If the input is a sequence $\mathbf{e}_k$ which is equal to 1 at time k and zero otherwise, then since $\mathbf{e}_k = D^k\mathbf{e}_0$, by time invariance the output must be $D^k\mathbf{g}$. Then since an arbitrary input sequence $\mathbf{u}$ can be written as $\mathbf{u} = \sum_k u_k \mathbf{e}_k$, by linearity the output must be the linear combination

$$\mathbf{y} = \sum_k u_k D^k \mathbf{g}.$$

The output y_k at time k is thus given by the convolution

$$y_k = \sum_{k' \le k} u_{k'} g_{k-k'}, \tag{9.1}$$

where we use the fact that by causality $g_{k-k'} = 0$ if $k < k'$. In other words, $\mathbf{y}$ is the convolution of the input sequence $\mathbf{u}$ and the impulse response $\mathbf{g}$:

$$\mathbf{y} = \mathbf{u} * \mathbf{g}.$$

It is important to note that the sum (**??**) that defines y_k is well defined if and only if it is a sum of only a finite number of nonzero terms, since over finite fields there is no notion of convergence of an infinite sum. The sum (**??**) is finite if and only if one of the following two conditions holds:

(a) There are only a finite number of nonzero elements g_k in the impulse response $\mathbf{g}$;

(b) For every k, there are only a finite number of nonzero elements $u_{k'}$ with $k' \leq k$ in the input sequence $\mathbf{u}$. This occurs if and only if there are only a finite number of nonzero elements u_k with negative time indices k. Such a sequence is called a *Laurent sequence*.

Since we do not in general want to restrict $\mathbf{g}$ to have a finite number of nonzero terms, we will henceforth impose the condition that the input sequence $\mathbf{u}$ must be Laurent, in order to guarantee that the sum (**??**) is well defined. With this condition, we have our desired result:

Theorem 9.1 (An LTI system is characterized by its impulse response) *If an LTI system over $\mathbb{F}_q$ has impulse response $\mathbf{g}$, then the output sequence in response to an arbitrary Laurent input sequence $\mathbf{u}$ is the convolution $\mathbf{y} = \mathbf{u} * \mathbf{g}$.*

9.1.2 The field of Laurent sequences

A nonzero Laurent sequence $\mathbf{u}$ has a definite starting time or *delay*, namely the time index of the first nonzero element: $\text{del}\ \mathbf{u} = \min\{k : u_k \neq 0\}$. The zero sequence $\mathbf{0}$ is Laurent, but has no definite starting time; by convention we define $\text{del}\ \mathbf{0} = \infty$. A *causal sequence* is a Laurent sequence with non-negative delay.

The (componentwise) sum of two Laurent sequences is Laurent, with delay not less than the minimum delay of the two sequences. The Laurent sequences form an abelian group under sequence addition, whose identity is the zero sequence $\mathbf{0}$. The additive inverse of a Laurent sequence $\mathbf{x}$ is $-\mathbf{x}$.

The convolution of two Laurent sequences is a well-defined Laurent sequence, whose delay is equal to the sum of the delays of the two sequences. The nonzero Laurent sequences form an abelian group under convolution, whose identity is the unit impulse $\mathbf{e}_0$. The inverse under convolution of a nonzero Laurent sequence $\mathbf{x}$ is a Laurent sequence $\mathbf{x}^{-1}$ which may be determined by long division, and which has delay equal to $\text{del}\ \mathbf{x}^{-1} = -\text{del}\ \mathbf{x}$.

Thus the set of all Laurent sequences forms a field under sequence addition and convolution.

9.1.3 D-transforms

The fact that the input/output map in any LTI system may be written as a convolution $\mathbf{y} = \mathbf{u} * \mathbf{g}$ suggests the use of polynomial-like notation, under which convolution becomes multiplication.

Therefore let us define the formal power series $u(D) = \sum_k u_k D^k$, $g(D) = \sum_k g_k D^k$, and $y(D) = \sum_k y_k D^k$. These are called "D-transforms," although the term "transform" may be misleading because these are still time-domain representations of the corresponding sequences.

In these expressions D is algebraically just an indeterminate (place-holder). However, D may also be regarded as representing the delay operator, because if the D-transform of $\mathbf{g}$ is $g(D)$, then the D-transform of $D\mathbf{g}$ is $Dg(D)$.

These expressions appear to be completely analogous to the "z-transforms" used in the theory of discrete-time real or complex LTI systems, with the substitution of D for z^{-1}. The subtle difference is that in the real or complex case z is often regarded as a complex number in the "frequency domain" and an expression such as $g(z^{-1})$ as a set of values as z ranges over $\mathbb{C}$; *i.e.,* as a true frequency-domain "transform."

It is easy to see that the convolution $\mathbf{y} = \mathbf{u} * \mathbf{g}$ then translates to

$$y(D) = u(D)g(D),$$

if for multiplication of D-transforms we use the usual rule of polynomial multiplication,

$$y_k = \sum_{k'} u_{k'} g_{k-k'},$$

since this expression is identical to (**??**). Briefly, convolution of sequences corresponds to multiplication of D-transforms.

In general, if $x(D)$ and $y(D)$ are D-transforms, then the product $x(D)y(D)$ is well defined when either $x(D)$ or $y(D)$ is finite, or when both $x(D)$ and $y(D)$ are Laurent. Since a causal impulse response $g(D)$ is Laurent, the product $u(D)g(D)$ is well defined whenever $u(D)$ is Laurent.

Addition of sequences is defined by componentwise addition of their components, as with vector addition. Correspondingly, addition of D-transforms is defined by componentwise addition. In other words, D-transform addition and multiplication are defined in the same way as polynomial addition and multiplication, and are consistent with the addition and convolution operations for the corresponding sequences.

It follows that the set of all Laurent D-transforms $x(D)$, which are called the Laurent power series in D over $\mathbb{F}_q$ and denoted by $\mathbb{F}_q((D))$, form a field under D-transform addition and multiplication, with additive identity 0 and multiplicative identity 1.

9.1.4 Categories of D-transforms

We pause to give a brief systematic exposition of various other categories of D-transforms.

Let $f(D) = \sum_{k \in \mathbb{Z}} f_k D^k$ be the D-transform of a sequence $\mathbf{f}$. We say that $\mathbf{f}$ or $f(D)$ is "zero on the past" (resp. "finite on the past") if it has zero (resp. a finite number) of nonzero f_k with negative time indices k, and "finite on the future" if it has a finite number of nonzero f_k with non-negative time indices k. $f(D)$ is *finite* if it is finite on both past and future.

(a) (Polynomials $\mathbb{F}_q[D]$.) If $f(D)$ is zero on the past and finite on the future, then $f(D)$ is a *polynomial* in D over $\mathbb{F}_q$. The set of all polynomials in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q[D]$. D-transform addition and multiplication of polynomials is the same as polynomial addition and multiplication. Under these operations, $\mathbb{F}_q[D]$ is a ring, and in fact an integral domain (see Chapter 7).

(b) (Formal power series $\mathbb{F}_q[[D]]$.) If $f(D)$ is zero on the past and unrestricted on the future, then $f(D)$ is a *formal power series* in D over $\mathbb{F}_q$. The set of all formal power series in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q[[D]]$. Under D-transform addition and multiplication, $\mathbb{F}_q[[D]]$ is a ring, and in fact an integral domain. However, $\mathbb{F}_q[[D]]$ is not a field, because D has no inverse in $\mathbb{F}_q[[D]]$. A formal power series corresponds to a causal sequence.

(c) (Laurent polynomials $\mathbb{F}_q[D, D^{-1}]$.) If $f(D)$ is finite, then $f(D)$ is a *Laurent polynomial* in D over $\mathbb{F}_q$. The set of all Laurent polynomials in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q[D, D^{-1}]$. Under D-transform addition and multiplication, $\mathbb{F}_q[D, D^{-1}]$ is a ring, and in fact an integral domain. However, $\mathbb{F}_q[D, D^{-1}]$ is not a field, because $1 + D$ has no inverse in $\mathbb{F}_q[D, D^{-1}]$. A Laurent polynomial corresponds to a finite sequence.

(d) (Laurent power series $\mathbb{F}_q((D))$.) If $f(D)$ is finite on the past and unrestricted on the future, then $f(D)$ is a *Laurent power series* in D over $\mathbb{F}_q$. The set of all Laurent power series in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q((D))$. As we have already seen, under D-transform addition and multiplication, $\mathbb{F}_q((D))$ is a field; *i.e.,* every nonzero $f(D) \in \mathbb{F}_q((D))$ has an inverse $f^{-1}(D) \in \mathbb{F}_q((D))$ such that $f(D)f^{-1}(D) = 1$, which can be found by long division of D-transforms; *e.g.,* over any field, the inverse of D is D^{-1}, and the inverse of $1 + D$ is $1 - D + D^2 - D^3 + \cdots$.

(e) (Bi-infinite power series $\mathbb{F}_q[[D, D^{-1}]]$.) If $f(D)$ is unrestricted on the past and future, then $f(D)$ is a *bi-infinite power series* in D over $\mathbb{F}_q$. The set of all bi-infinite power series in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q[[D, D^{-1}]]$. As we have seen, D-transform multiplication is not well defined for all $f(D), g(D) \in \mathbb{F}_q[[D, D^{-1}]]$, so $\mathbb{F}_q[[D, D^{-1}]]$ is merely a group under D-transform addition.

(f) (Rational functions $\mathbb{F}_q(D)$.) A Laurent power series $f(D)$ (or the corresponding sequence $\mathbf{f}$) is called *rational* if it can be written as $f(D) = n(D)/d(D)$, where $n(D)$ and $d(D) \neq 0$ are polynomials in $\mathbb{F}_q[D]$, and $n(D)/d(D)$ denotes the product of $n(D)$ with $d^{-1}(D)$. The set of all rational power series in D over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q(D)$. It is easy to verify that that $\mathbb{F}_q(D)$ is closed under D-transform addition and multiplication. Moreover, the multiplicative inverse of a nonzero rational D-transform $f(D) = n(D)/d(D)$ is $f^{-1}(D) = d(D)/n(D)$, which is evidently rational. It follows that $\mathbb{F}_q(D)$ is a field.

It is easy to see that $\mathbb{F}_q[D] \subset \mathbb{F}_q[D, D^{-1}] \subset \mathbb{F}_q(D) \subset \mathbb{F}_q((D))$, since every polynomial is a Laurent polynomial, and every Laurent polynomial is rational (for example, $D^{-1} + 1 = (1 + D)/D$). The rational functions and the Laurent power series have the nicest algebraic properties, since both are fields. Indeed, the rational functions form a subfield of the Laurent power series, as the rational numbers $\mathbb{Q}$ form a subfield of the real numbers $\mathbb{R}$. The polynomials form a subring of the rational functions, as the integers $\mathbb{Z}$ form a subring of $\mathbb{Q}$.

The following exercise shows that an infinite Laurent D-transform $f(D)$ is rational if and only if the corresponding sequence $\mathbf{f}$ eventually becomes periodic. (This should remind the reader of the fact that a real number is rational if and only if its decimal expansion is eventually periodic.)

Exercise 1 (rational = eventually periodic). Show that a Laurent D-transform $f(D)$ is rational if and only if the corresponding sequence $\mathbf{f}$ is finite or eventually becomes periodic. [Hints: (a) show that a sequence $\mathbf{f}$ is eventually periodic with period P if and only if its D-transform $f(D)$ can be written as $f(D) = g(D)/(1 - D^P)$, where $g(D)$ is a Laurent polynomial; (b) using the results of Chapter 7, show that every nonzero polynomial $d(D) \in \mathbb{F}_q[D]$ divides $1 - D^P$ for some integer P.] □

9.1.5 Realizations of LTI systems

So far we have characterized an LTI system over $\mathbb{F}_q$ by its input/output map, which we have shown is entirely determined by its impulse response $\mathbf{g}$ or the corresponding D-transform $g(D)$. The only restriction that we have placed on the impulse response is that it be causal; *i.e.,* that $g(D)$ be a formal power series in $\mathbb{F}_q[[D]]$. In order that the input/output map be well-defined, we have further required that the input $u(D)$ be Laurent, $u(D) \in \mathbb{F}_q(D)$.

In this subsection we consider realizations of such an LTI system. A realization is a block diagram whose blocks represent $\mathbb{F}_q$-adders, $\mathbb{F}_q$-multipliers, and $\mathbb{F}_q$-delay (memory) elements, which we take as our elementary LTI systems. An $\mathbb{F}_q$-adder may have any number of inputs in $\mathbb{F}_q$, and its output is their (instantaneous) sum. An $\mathbb{F}_q$-multiplier may have any number of inputs in $\mathbb{F}_q$, and its output is their (instantaneous) product. An $\mathbb{F}_q$-delay element has a single input in $\mathbb{F}_q$, and a single output which is equal to the input one time unit earlier.

For example, if an LTI system has impulse response $g(D) = 1 + \alpha D + \beta D^3$, then it can be realized by the realization shown in Figure 2. In this realization there are three delay (memory) elements, arranged as a shift register of length 3. It is easy to check that the input/output map is given by $y(D) = u(D) + \alpha D u(D) + \beta D^3 u(D) = u(D)g(D)$.

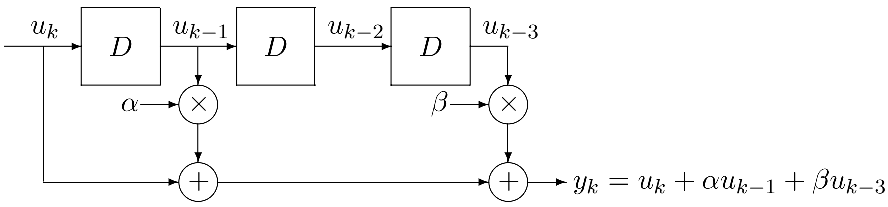

Figure 2. Realization of an LTI system with impulse response $g(D) = 1 + \alpha D + \beta D^3$.

More generally, it is easy to see that if $g(D)$ is finite (polynomial) with degree $\deg g(D) = \nu$, then an LTI system with impulse response $g(D)$ can be realized similarly, using a shift register of length ν.

The *state* of a realization at time k is the set of contents of its memory elements. For example, in Figure 2 the state at time k is the 3-tuple $(u_{k-1}, u_{k-2}, u_{k-3})$. The *state space* is the set of all possible states. If a realization over $\mathbb{F}_q$ has only a finite number ν of memory elements, then the state space has size q^ν, which is finite. For example, the state space size in Figure 2 is q^3.

Now suppose that we consider only realizations with a finite number of blocks; in particular, with a finite number of memory elements, and thus a finite state space size. What is the most general impulse response that we can realize? If the input is the impulse $e_0(D) = 1$, then the input is zero after time zero, so the impulse response is determined by the autonomous (zero-input) behavior after time zero. Since the state space size is finite, it is clear that the autonomous behavior must be periodic after time zero, and therefore the impulse response must be eventually periodic— *i.e.,* rational. We conclude that finite realizations can realize only rational impulse responses.

Conversely, given a rational impulse response $g(D) = n(D)/d(D)$, where $n(D)$ and $d(D) \neq 0$ are polynomial, it is straightforward to show that $g(D)$ can be realized with a finite number of memory elements, and in fact with $\nu = \max\{\deg n(D), \deg d(D)\}$ memory elements. For example, Figure 3 shows a realization of an LTI system with the impulse response $g(D) =$

$(1+\alpha D+\beta D^3)/(1-D^2-D^3)$ using $\nu=3$ memory elements. Because the impulse response is infinite, the realization necessarily involves feedback, in contrast to a feedbackfree realization of a finite impulse response as in Figure 2.

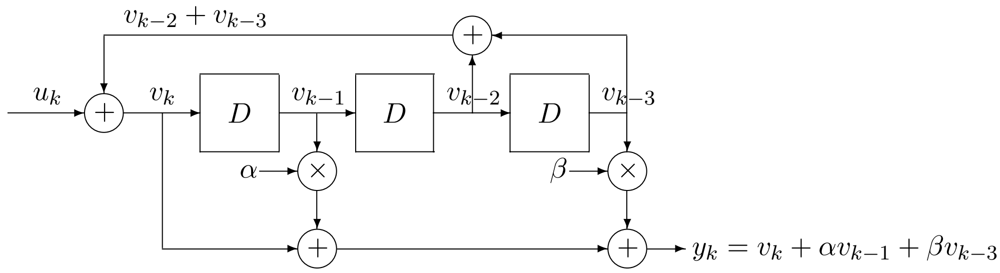

Figure 3. Realization of an LTI system with impulse response
$g(D) = (1+\alpha D+\beta D^3)/(1-D^2-D^3)$.

Exercise 2 (rational realizations). Generalize Figure 2 to realize any rational impulse response $g(D) = n(D)/d(D)$ with $\nu = \max\{\deg n(D), \deg d(D)\}$ memory elements. □

In summary:

Theorem 9.2 (finitely realizable = rational) *An LTI system with causal (thus Laurent) impulse response $g(D)$ has a finite realization if and only if $g(D)$ is rational. If $g(D) = n(D)/d(D)$, then there exists a realization with state space size q^ν, where $\nu = \max\{\deg n(D), \deg d(D)\}$. The realization can be feedbackfree if and only if $g(D)$ is polynomial.*

9.2 Rate-$1/n$ binary linear convolutional codes

A *rate-1/n binary linear convolutional encoder* is a single-input, n-output LTI system over the binary field $\mathbb{F}_2$. Such a system is characterized by the n impulse responses $\{g_j(D), 1 \le j \le n\}$, which can be written as an n-tuple $\mathbf{g}(D) = (g_1(D), \dots, g_n(D))$.

If the input sequence $u(D)$ is Laurent, then the n output sequences $\{y_j(D),\ 1 \le j \le n\}$ are well defined and are given by $y_j(D) = u(D)g_j(D)$ More briefly, the output n-tuple $\mathbf{y}(D) = (y_1(D), \dots, y_n(D))$ is given by $\mathbf{y}(D) = u(D)\mathbf{g}(D)$.

The encoder is *polynomial* (or "feedforward") if all impulse responses $g_j(D)$ are polynomial. In that case there is a shift-register realization of $\mathbf{g}(D)$ as in Figure 1 involving a single shift register of length $\nu = \deg \mathbf{g}(D) = \max_j \deg g_j(D)$. The encoder state space size is then 2^ν.

Example 1 (Rate-1/2 convolutional encoder). A rate-1/2 polynomial binary convolutional encoder is defined by a polynomial 2-tuple $\mathbf{g}(D) = (g_1(D), g_2(D))$. A shift-register realization of $\mathbf{g}(D)$ involves a single shift register of length $\nu = \max\{\deg g_1(D), \deg g_2(D)\}$ and has a state space of size 2^ν. For example, the impulse response 2-tuple $\mathbf{g}(D) = (1+D^2, 1+D+D^2)$ is realized by the 4-state rate-1/2 encoder illustrated in Figure 1. □

More generally, the encoder is *realizable* if all impulse responses $g_j(D)$ are causal and rational, since each response must be (finitely) realizable by itself, and we can obtain a finite realization of all of them by simply realizing each one separately.

A more efficient (and in fact minimal, although we will not show this yet) realization may be obtained as follows. If each $g_j(D)$ is causal and rational, then $g_j(D) = n_j(D)/d_j(D)$ for polynomials $n_j(D)$ and $d_j(D) \neq 0$, where by reducing to lowest terms, we may assume that $n_j(D)$ and $d_j(D)$ have no common factors. Then we can write

$$\mathbf{g}(D) = \frac{(n'_1(D), n'_2(D), \ldots, n'_n(D))}{d(D)} = \frac{\mathbf{n}'(D)}{d(D)},$$

where the common denominator polynomial $d(D)$ is the least common multiple of the denominator polynomials $d_j(D)$, and $\mathbf{n}'(D)$ and $d(D)$ have no common factors. In order that $\mathbf{g}(D)$ be causal, $d(D)$ cannot be divisible by D; *i.e.,* $d_0 = 1$.

Now, as the reader may verify by extending Exercise 2, this set of n impulse responses may be realized by a single shift register of length $\nu = \max\{\deg \mathbf{n}'(D), \deg d(D)\}$ memory elements, with feedback coefficients determined by the common denominator polynomial $d(D)$ as in Figure 3, and with the n outputs formed as n different linear combinations of the shift register contents, as in Figure 1. To summarize:

Theorem 9.3 (Rate-$1/n$ convolutional encoders) *If $\mathbf{g}(D) = (g_1(D), g_2(D), \ldots, g_n(D))$ is the set of n impulse responses of a rate-$1/n$ binary linear convolutional encoder, then there exists a unique denominator polynomial $d(D)$ with $d_0 = 1$ such that we can write each $g_j(D)$ as $g_j(D) = n_j(D)/d(D)$, where the numerator polynomials $n_j(D)$ and $d(D)$ have no common factor. There exists a (minimal) realization of $\mathbf{g}(D)$ with $\nu = \max\{\deg \mathbf{n}'(D), \deg d(D)\}$ memory elements and thus 2^ν states, which is feedbackfree if and only if $d(D) = 1$.*

9.2.1 Finite-state representations of convolutional codes

Because a convolutional encoder is a finite-state machine, it may be characterized by a finite *state-transition diagram.* For example, the encoder of Example 1 has the 4-state state-transition diagram shown in Figure 4(a). Each state is labelled by two bits representing the contents of the shift register, and each state transition is labelled by the two output bits associated with that transition.

Alternatively, a finite-state encoder may be characterized by a *trellis diagram*, which is simply a state-transition diagram with the states s_k at each time k shown separately. Thus there are transitions only from states s_k at time k to states s_{k+1} at time $k+1$. Figure 4(b) shows a segment of a trellis diagram for the encoder of Example 1, with states labelled as in Figure 4(a).

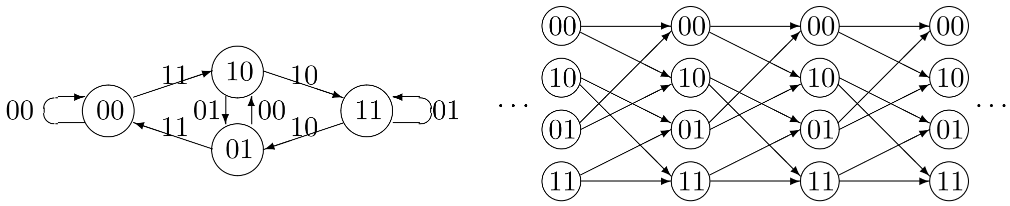

Figure 4. (a) Four-state state transition diagram; (b) Corresponding trellis diagram.

9.2.2 Rate-$1/n$ binary linear convolutional codes

A rate-$1/n$ binary linear convolutional code $\mathcal{C}$ is defined as the set of "all" output n-tuples $\mathbf{y}(D)$ that can be generated by a rate-$1/n$ binary linear convolutional encoder. If the encoder is characterized by an impulse response n-tuple $\mathbf{g}(D)$, then $\mathcal{C}$ is the set of output n-tuples $\mathbf{y}(D) = u(D)\mathbf{g}(D)$ as the input sequence $u(D)$ ranges over "all" possible input sequences.

How to define the set of "all" input sequences is actually a subtle question. We have seen that if $\mathbf{g}(D)$ is not polynomial, then we must restrict the input sequences $u(D)$ to be Laurent in order that the output $\mathbf{y}(D)$ be well-defined. On the other hand, $u(D)$ should be permitted to be infinite, because it can happen that some finite code sequences are generated by infinite input sequences (this phenomenon is called catastrophicity; see below).

The following definitions of the set of "all" input sequences meet both these criteria and are therefore OK; all have been used in the literature.

- The set $\mathbb{F}_2((D))$ of all formal Laurent series;
- The set $\mathbb{F}_2(D)$ of all rational functions;
- The set $\mathbb{F}_2[[D]]$ of all formal power series.

Here we will use the set $\mathbb{F}_2((D))$ of formal Laurent series. As we have seen, $\mathbb{F}_2((D))$ is a field under D-transform addition and multiplication, and includes the field $\mathbb{F}_2(D)$ of rational functions as a proper subfield. Since $\mathbf{g}(D)$ is rational, the set $\mathbb{F}_2(D)$ would suffice to generate all finite code sequences; however, we prefer $\mathbb{F}_2((D))$ because there seems no reason to constrain input sequences to be rational. We prefer either of these to $\mathbb{F}_2[[D]]$ because $\mathbb{F}_2((D))$ and $\mathbb{F}_2(D)$ are time-invariant, whereas $\mathbb{F}_2[[D]]$ is not; moreover, $\mathbb{F}_2[[D]]$ is not a field, but only a ring.

The *convolutional code* generated by $\mathbf{g}(D) \in (\mathbb{F}_2(D))^n$ will therefore be defined as

$$\mathcal{C} = \{\mathbf{y}(D) = u(D)\mathbf{g}(D), u(D) \in \mathbb{F}_2((D))\}.$$

The *rational subcode* of $\mathcal{C}$ is the set of all its rational sequences,

$$\mathcal{C}_r = \{\mathbf{y}(D) \in \mathcal{C} : \mathbf{y}(D) \in (\mathbb{F}_2(D))^n\},$$

and the *finite subcode* of $\mathcal{C}$ is the set of all its Laurent polynomial sequences,

$$\mathcal{C}_f = \{\mathbf{y}(D) \in \mathcal{C} : \mathbf{y}(D) \in (\mathbb{F}_2[D, D^{-1}])^n\}.$$

Since $\mathbb{F}_2[D, D^{-1}] \subset \mathbb{F}_2(D) \subset \mathbb{F}_2((D))$, we have $\mathcal{C}_f \subset \mathcal{C}_r \subset \mathcal{C}$.

Exercise 3 (input/output properties)

(a) Show that $\mathbf{y}(D)$ is an n-tuple of formal Laurent series, $\mathbf{y}(D) \in (\mathbb{F}_2((D)))^n$.

(b) Show that $\mathbf{y}(D)$ is rational if and only if $u(D)$ is rational; *i.e.,*

$$\mathcal{C}_r = \{\mathbf{y}(D) = u(D)\mathbf{g}(D), u(D) \in \mathbb{F}_2(D)\}.$$

(c) Show that $\mathbf{y}(D)$ is finite if and only if $u(D) = a(D)\mathrm{lcm}\{d_j(D)\}/\gcd\{n_j(D)\}$, where $a(D)$ is finite, $\mathrm{lcm}\{d_j(D)\}$ is the least common multiple of the denominators $d_j(D)$ of the $g_j(D)$, and $\gcd\{n_j(D)\}$ is the greatest common divisor of their numerators. □

A convolutional code $\mathcal{C}$ has the group property under D-transform addition, since if $\mathbf{y}(D) = u(D)\mathbf{g}(D)$ and $\mathbf{y}'(D) = u'(D)\mathbf{g}(D)$ are any two convolutional code n-tuples generated by two input sequences and $u(D)$ and $u'(D)$, respectively, then the n-tuple $\mathbf{y}(D) + \mathbf{y}'(D)$ is generated by the input sequence $u(D) + u'(D)$. It follows that $\mathcal{C}$ is a vector space over the binary field $\mathbb{F}_2$, an infinite-dimensional subspace of the infinite-dimensional vector space $(\mathbb{F}_2((D)))^n$ of all Laurent n-tuples.

At a higher level, a rate-$1/n$ convolutional code $\mathcal{C}$ is a one-dimensional subspace with *generator* $\mathbf{g}(D)$ of the n-dimensional vector space $(\mathbb{F}_2((D)))^n$ of all Laurent n-tuples over the Laurent field $\mathbb{F}_2((D))$. Similarly, the rational subcode $\mathcal{C}_r$ is a one-dimensional subspace with generator $\mathbf{g}(D)$ of the n-dimensional vector space $(\mathbb{F}_2(D))^n$ of all rational n-tuples over the rational field $\mathbb{F}_2(D)$. In these respects rate-$1/n$ convolutional codes are like $(n, 1)$ linear block codes.

9.2.3 Encoder equivalence

Two generator n-tuples $\mathbf{g}(D)$ and $\mathbf{g}'(D)$ will now be defined to be *equivalent* if they generate the same code, $\mathcal{C} = \mathcal{C}'$. We will shortly seek the best encoder to generate any given code $\mathcal{C}$.

Theorem 9.4 (Rate-$1/n$ encoder equivalence) *Two generator n-tuples $\mathbf{g}(D), \mathbf{g}'(D)$ are equivalent if and only if $\mathbf{g}(D) = u(D)\mathbf{g}'(D)$ for some nonzero rational function $u(D) \in \mathbb{F}_2(D)$.*

Proof. If the two encoders generate the same code, then $\mathbf{g}(D)$ must be a sequence in the code generated by $\mathbf{g}'(D)$, so we have $\mathbf{g}(D) = u(D)\mathbf{g}'(D)$ for some nonzero $u(D) \in \mathbb{F}_2((D))$. Moreover $\mathbf{g}(D)$ is rational, so from Exercise 3(b) $u(D)$ must be rational. Conversely, if $\mathbf{g}(D) = u(D)\mathbf{g}'(D)$ and $\mathbf{y}(D) \in \mathcal{C}$, then $\mathbf{y}(D) = v(D)\mathbf{g}(D)$ for some $v(D) \in F((D))$; thus $\mathbf{y}(D) = v(D)u(D)\mathbf{g}'(D)$ and $\mathbf{y}(D) \in \mathcal{C}'$, so $\mathcal{C} \subseteq \mathcal{C}'$. Since $\mathbf{g}'(D) = \mathbf{g}'(D)/u(D)$, a similar argument can be used in the other direction to show $\mathcal{C}' \subseteq \mathcal{C}$, so we can conclude $\mathcal{C} = \mathcal{C}'$. □

Thus let $\mathbf{g}(D) = (g_1(D), g_2(D), \ldots, g_n(D))$ be an arbitrary rational generator n-tuple. We can obtain an equivalent polynomial generator n-tuple by multiplying $\mathbf{g}(D)$ by any polynomial which is a multiple of all denominator polynomials $d_j(D)$, and in particular by their least common multiple $\mathrm{lcm}\{d_j(D)\}$. Thus we have:

Corollary 9.5 *Every generator n-tuple $\mathbf{g}(D)$ is equivalent to a polynomial n-tuple $\mathbf{g}'(D)$.*

A generator n-tuple $\mathbf{g}(D)$ is called *delay-free* if at least one generator has a nonzero term at time index 0; *i.e.,* if $\mathbf{g} \neq \mathbf{0}$. If $\mathbf{g}(D)$ is not delay-free, then we can eliminate the delay by using instead the equivalent generator n-tuple $\mathbf{g}'(D) = \mathbf{g}(D)/D^{\mathrm{del}\ \mathbf{g}(D)}$, where $\mathrm{del}\ \mathbf{g}(D)$ is the smallest time index k such that $\mathbf{g}_k \neq \mathbf{0}$. Thus:

Corollary 9.6 *Every generator n-tuple $\mathbf{g}(D)$ is equivalent to a delay-free n-tuple $\mathbf{g}'(D)$, namely $\mathbf{g}'(D) = \mathbf{g}(D)/D^{\mathrm{del}\ \mathbf{g}(D)}$.*

A generator n-tuple $\mathbf{g}(D)$ is called *catastrophic* if there exists an infinite input sequence $u(D)$ that generates a finite output n-tuple $\mathbf{y}(D) = u(D)\mathbf{g}(D)$. Any realization of $\mathbf{g}(D)$ must thus have a cycle in its state-transition diagram other than the zero-state self-loop such that the outputs are all zero during the cycle.

There is a one-to-one correspondence between the set of all paths through the trellis diagram of a convolutional encoder and the set $\{u(D)\}$ of all input sequences. However, the correspondence to the set of all output sequences $\{u(D)\mathbf{g}(D)\}$— *i.e.,* to the convolutional code $\mathcal{C}$ generated by the encoder— could be many-to-one. For example, if the encoder is catastrophic, then there are at least two paths corresponding to the all-zero output sequence, and from the group property of the encoder there will be at least two paths corresponding to all output sequences. In fact, the correspondence between convolutional code sequences and trellis paths is one-to-one if and only if $\mathbf{g}(D)$ is noncatastrophic.

By Exercise 3(c), $\mathbf{y}(D)$ is finite if and only if $u(D) = a(D)\mathrm{lcm}\{d_j(D)\}/\gcd\{n_j(D)\}$ for some finite $a(D)$. Since $\mathrm{lcm}\{d_j(D)\}$ is also finite, $u(D)$ can be infinite if and only if $\gcd\{n_j(D)\}$ has an infinite inverse. This is true if and only $\gcd\{n_j(D)\} \neq D^d$ for some integer d. In summary:

Theorem 9.7 (Catastrophicity) *A generator n-tuple $\mathbf{g}(D)$ is catastrophic if and only if all numerators $n_j(D)$ have a common factor other than D.*

Example 2. The rate-1/2 encoder $\mathbf{g}(D) = (1 + D^2, 1 + D + D^2)$ is noncatastrophic, since the polynomials $1 + D^2$ and $1 + D + D^2$ are relatively prime. However, the rate-1/2 encoder $\mathbf{g}(D) = (1 + D^2, 1 + D)$ is catastrophic, since $1 + D^2$ and $1 + D$ have the common divisor $1 + D$; thus the infinite input $1/(1 + D) = 1 + D + D^2 + \ldots$ leads to the finite output $\mathbf{y}(D) = (1 + D, 1)$ (due to a self-loop from the 11 state to itself with output 00). □

If $\mathbf{g}(D)$ is catastrophic with greatest common factor $\gcd\{n_j(D)\}$, then we can easily find an equivalent noncatastrophic generator n-tuple by dividing out the common factor: $\mathbf{g}'(D) = \mathbf{g}(D)/\gcd\{n_j(D)\}$. For instance, in Example 2, we should replace $(1 + D^2, 1 + D)$ by $(1 + D, 1)$.

In summary, we can and should eliminate denominators and common numerator factors:

Corollary 9.8 *Every generator n-tuple $\mathbf{g}(D)$ is equivalent to a unique (up to a scalar multiple) noncatastrophic delay-free polynomial n-tuple $\mathbf{g}'(D)$, namely*

$$\mathbf{g}'(D) = \frac{\mathrm{lcm}\{d_j(D)\}}{\gcd\{n_j(D)\}}\mathbf{g}(D).$$

Thus every rate-$1/n$ binary convolutional cade $\mathcal{C}$ has a unique noncatastrophic delay-free polynomial generator n-tuple $\mathbf{g}(D)$, which is called *canonical.* Canonical generators have many nice properties, including:

- The finite subcode $\mathcal{C}_f$ is the subcode generated by the finite input sequences.
- The feedbackfree shift-register realization of $\mathbf{g}(D)$ is minimal over all equivalent encoders.

In general, it has been shown that a convolutional encoder is *minimal*— that is, has the minimal number of states over all encoders for the same code— if and only if it is noncatastrophic and moreover there are no state transitions to or from the zero state with all-zero outputs, other than from the zero state to itself. For a rate-$1/n$ encoder with generator $\mathbf{g}(D) = (n_1(D), \ldots, n_n(D))/d(D)$, this means that the numerator polynomials $n_j(D)$ must have no common factor, and that the degree of the denominator polynomial $d(D)$ must be no greater than the maximum degree of the numerators.

A *systematic* encoder for a rate-$1/n$ convolutional code is one in which the input sequence appears unchanged in one of the n output sequences. A systematic encoder for a code generated by polynomial encoder $\mathbf{g}(D) = (g_1(D), \ldots, g_n(D))$ may be obtained by multiplying all generators by $1/g_1(D)$ (provided that $g_{10} = 1$). For example, $\mathbf{g}(D) = (1, (1 + D + D^2)/(1 + D^2))$ is a systematic encoder for the code generated by $(1 + D^2, 1 + D + D^2)$.

A systematic encoder is always delay-free, noncatastrophic and minimal. Sometimes systematic encoders are taken as an alternative class of nonpolynomial canonical encoders.

9.2.4 Algebraic theory of rate-k/n convolutional codes

There is a more general theory of rate-k/n convolutional codes, the main result of which is as follows. Let $\mathcal{C}$ be any time-invariant subspace of the vector space of all $\mathbf{y}(D) \in (\mathbb{F}_q((D)))^n$. A *canonical polynomial encoder* for $\mathcal{C}$ may be constructed by the following greedy algorithm:

Initialization:	set $i = 0$ and $\mathcal{C}_0 = \{\mathbf{0}\}$;
Do loop:	if $\mathcal{C}_i = \mathcal{C}$, we are done, and $k = i$;
	otherwise, increase i by 1 and take any polynomial $\mathbf{y}(D)$ of
	least degree in $\mathcal{C} \setminus \mathcal{C}_i$ as $\mathbf{g}_i(D)$.

The shift-register realization of the resulting $k \times n$ generator matrix $G(D) = \{\mathbf{g}_i(D), 1 \le i \le k\}$ is then a minimal encoder for $\mathcal{C}$, in the sense that no other encoder for $\mathcal{C}$ has fewer memory elements. The degrees $\nu_i = \deg \mathbf{g}_i(D) = \max_{1 \le j \le n} \deg g_{ij}(D)$, $1 \le i \le k$, are uniquely determined by this construction, and are called the "constraint lengths" (or controllability indices) of $\mathcal{C}$; their maximum $\nu_{\max}$ is the (controller) memory of $\mathcal{C}$, and their sum $\nu = \sum_i \nu_i$ (the "overall constraint length") determines the size 2^ν of the state space of any minimal encoder for $\mathcal{C}$. Indeed, this encoder turns out to have every property one could desire, except for systematicity.

9.3 Terminated convolutional codes

A rate-$1/n$ *terminated convolutional code* $\mathcal{C}_\mu$ is the code generated by a polynomial convolutional encoder $\mathbf{g}(D)$ (possibly catastrophic) when the input sequence $u(D)$ is constrained to be a polynomial of degree less than some nonnegative integer μ:

$$\mathcal{C}_\mu = \{u(D)\mathbf{g}(D) \mid \deg u_i(D) < \mu\}.$$

In this case the total number of possibly nonzero input bits is $k = \mu$, and the total number of possibly nonzero output bits is $n' = n(\mu + \nu)$, where $\nu = \max \deg g_j(D)$ is the shift-register length in a shift-register realization of $\mathbf{g}(D)$. Thus a terminated convolutional code may be regarded as an $(n', k) = (n(\mu + \nu), \mu)$ binary linear block code.

The trellis diagram of a terminated convolutional code is finite. For example, Figure 5 shows the trellis diagram of the 4-state ($\nu = 2$) rate-1/2 encoder of Example 1 when $\mu = 5$, which yields a $(14, 5, 5)$ binary linear block code.

Exercise 4. Show that if the (catastrophic) rate-1/1 binary linear convolutional code generated by $g(D) = 1 + D$ is terminated with $\deg u(D) < \mu$, then the resulting code is a $(\mu + 1, \mu, 2)$ SPC code. Conclude that any binary linear SPC code may be represented by a 2-state trellis diagram. □

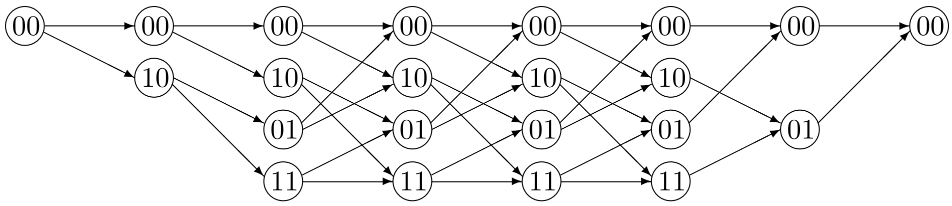

Figure 5. Trellis diagram of a terminated rate-1/2 convolutional encoder.

Exercise 5. Show that if the (catastrophic) rate-1/1 binary linear convolutional code generated by $g(D) = 1 + D + D^3$ is terminated with $\mu = 4$, then the resulting code is a $(7, 4, 3)$ Hamming code. □

9.4 Maximum likelihood sequence detection: The VA

A convolutional code sequence $\mathbf{y}(D) = u(D)\mathbf{g}(D)$ may be transmitted through an AWGN channel in the usual fashion, by mapping each bit $y_{jk} \in \{0, 1\}$ to $s(y_{jk}) \in \{\pm\alpha\}$ via the usual 2-PAM map. We denote the resulting sequence by $s(\mathbf{y}(D))$. At the output of the channel, the received sequence is

$$\mathbf{r}(D) = s(\mathbf{y}(D)) + \mathbf{n}(D),$$

where $\mathbf{n}(D)$ is an iid Gaussian noise sequence with variance $N_0/2$ per dimension.

As usual, maximum-likelihood sequence detection is equivalent to minimum-distance (MD) detection; *i.e.,* find the code sequence $\mathbf{y}(D)$ such that $\|\mathbf{r}(D) - s(\mathbf{y}(D))\|^2$ is minimum. In turn, since $s(\mathbf{y}(D))$ is binary, this is equivalent to maximum-inner-product (MIP) detection: find the code sequence $\mathbf{y}(D)$ that maximizes

$$\langle \mathbf{r}(D), s(\mathbf{y}(D)) \rangle = \sum_k \langle \mathbf{r}_k, s(\mathbf{y}_k) \rangle = \sum_k \sum_j r_{kj} s(y_{kj}).$$

It may not be immediately clear that this is a well-posed problem, since all sequences are bi-infinite. However, we will now develop a recursive algorithm that solves this problem for terminated convolutional codes; it will then be clear that the algorithm continues to specify a well-defined maximum-likelihood sequence detector as the length of the terminated convolutional code goes to infinity.

The algorithm is the celebrated *Viterbi algorithm* (VA), which some readers may recognize simply as "dynamic programming" applied to a trellis diagram.

The first observation is that if we assign to each branch in the trellis a "metric" equal to $\|\mathbf{r}_k - s(\mathbf{y}_k)\|^2$, or equivalently $-\langle \mathbf{r}_k, s(\mathbf{y}_k) \rangle$, where $\mathbf{y}_k$ is the output n-tuple associated with that branch, then since each path through the trellis corresponds to a unique code sequence, MD or MIP detection becomes equivalent to finding the least-metric ("shortest") path through the trellis diagram.

The second observation is that the initial segment of the shortest path, say from time 0 through time k, must be the shortest path to whatever state s_k it passes through at time k, since if there were any shorter path to s_k, it could be substituted for that initial segment to create a shorter path overall, contradiction. Therefore it suffices at time k to determine and retain for each state

s_k at time k only the shortest path from the unique state at time 0 to that state, called the "survivor."

The last observation is that the time-$(k+1)$ survivors may be determined from the time-k survivors by the following recursive "add-compare-select" operation:

(a) For each branch from a state at time k to a state at time $k+1$, add the metric of that branch to the metric of the time-k survivor to get a candidate path metric at time $k+1$;

(b) For each state at time $k+1$, compare the candidate path metrics arriving at that state and select the path corresponding to the smallest as the survivor. Store the survivor path history from time 0 and its metric.

At the final time $\mu + \nu$, there is a unique state, whose survivor must be the (or at least a) shortest path through the trellis diagram of the terminated code.

The Viterbi algorithm has a regular recursive structure that is attractive for software or hardware implementation. Its complexity is clearly proportional to the number of trellis branches per unit of time ("branch complexity"), which in the case of a binary rate-k/n 2^ν-state convolutional code is $2^{k+\nu}$.

For infinite trellises, there are a few additional issues that must be addressed, but none lead to problems in practice.

First, there is the issue of how to get started. In practice, if the algorithm is simply initiated with arbitrary accumulated path metrics at each state, say all zero, then it will automatically synchronize in a few constraint lengths.

Second, path histories need to be truncated and decisions put out after some finite delay. In practice, it has been found empirically that with high probability all survivors will have the same history prior to about five constraint lengths in the past, so that a delay of this magnitude usually suffices to make the additional probability of error due to premature decisions negligible, for any reasonable method of making final decisions.

In some applications, it is important that the decoded path be a true path in the trellis; in this case, when decisions are made it is important to purge all paths that are inconsistent with those decisions.

Finally, the path metrics must be renormalized from time to time so that their magnitude does not become too great; this can be done by subtracting the minimum metric from all of them.

In summary, the VA works perfectly well on unterminated convolutional codes. Maximum-likelihood sequence detection (MLSD) of unterminated codes may be operationally defined by the VA.

More generally, the VA is a general method for maximum-likelihood detection of the state sequence of any finite-state Markov process observed in memoryless noise. For example, it can be used for maximum-likelihood sequence detection of digital sequences in the presence of intersymbol interference.

9.5 Performance analysis of convolutional codes

The performance analysis of convolutional codes is based on the notion of an "error event."

Suppose that the transmitted code sequence is $\mathbf{y}(D)$ and the detected code sequence is $\mathbf{y}'(D)$. Each of these sequences specifies a unique path through a minimal code trellis. Typically these paths will agree for long periods of time, but will disagree over certain finite intervals. An error event corresponds to one of these finite intervals. It begins when the path $\mathbf{y}'(D)$ first diverges from the path $\mathbf{y}(D)$, and ends when these two paths merge again. The error sequence is the difference $\mathbf{e}(D) = \mathbf{y}'(D) - \mathbf{y}(D)$ over this interval.

By the group property of a convolutional code $\mathcal{C}$, such an error sequence $\mathbf{e}(D)$ is a finite code sequence in $\mathcal{C}$. If the encoder $\mathbf{g}(D)$ is noncatastrophic (or if the code is a terminated code), then $\mathbf{e}(D) = u_e(D)\mathbf{g}(D)$ for some finite input sequence $u_e(D)$.

The probability of any such finite error event in AWGN is given as usual by

$$\Pr(\mathbf{y}'(D) \mid \mathbf{y}(D)) = Q^{\surd}(\|s(\mathbf{y}'(D)) - s(\mathbf{y}(D))\|^2/2N_0).$$

As in the block code case, we have

$$\|s(\mathbf{y}'(D)) - s(\mathbf{y}(D))\|^2 = 4\alpha^2 d_H(\mathbf{y}', \mathbf{y}).$$

The minimum error event probability is therefore governed by the minimum Hamming distance between code sequences $\mathbf{y}(D)$. For convolutional codes, this is called the *free distance* d_{free}.

By the group property of a convolutional code, d_{free} is equal to the minimum Hamming weight of any finite code sequence $\mathbf{y}(D)$; *i.e.,* a code sequence that starts and ends with semi-infinite all-zero sequences. In a minimal trellis, such a code sequence must start and end with semi-infinite all-zero state sequences. Thus d_{free} is simply the minimum weight of a trellis path that starts and ends in the zero state in a minimal trellis, which can easily be found by a search through the trellis using a version of the VA.

Example 1 (cont.) From the state-transition diagram of Figure 4(a) or the trellis diagram of Figure 4(b), it is clear that the the free distance of the example 4-state rate-1/2 code is $d_{\mathrm{free}} = 5$, and that the only code sequences with this weight are the generator sequence $\mathbf{g}(D) = (1 + D^2, 1 + D + D^2)$ and its time shifts $D^k\mathbf{g}(D)$. □

For a rate-k/n binary convolutional code, it makes sense to normalize the error probability per unit time of the code— *i.e.,* per k input or n output bits. The probability of an error event starting at a given time, assuming that no error event is already in progress at that time, may be estimated by the union bound estimate as

$$\begin{aligned} P_c(E) &\approx K_{\min}(\mathcal{C})Q^{\surd}(2\alpha^2 d_{\mathrm{free}}/N_0) &(9.2)\\ &= K_{\min}(\mathcal{C})Q^{\surd}(\gamma_c(\mathcal{C})(2E_b/N_0)), &(9.3)\end{aligned}$$

where $E_b = n\alpha^2/k$ is the average energy per bit, and

$$\gamma_c(\mathcal{C}) = d_{\mathrm{free}}(k/n) \qquad (9.4)$$

is the *nominal coding gain* of the convolutional code $\mathcal{C}$, while $K_{\min}(\mathcal{C})$ is the number of error events of weight d_{free} in $\mathcal{C}$ per unit time. Note that the nominal coding gain $\gamma_c(\mathcal{C}) = d_{\mathrm{free}}(k/n)$

is defined analogously to the block code case. Note also that for a time-invariant code $\mathcal{C}$, the error event probability $P_c(E)$ is independent of time.

For direct comparison to block codes, the error probability per bit may be estimated as:

$$P_b(E) = P_c(E)/k \approx K_b(\mathcal{C}) Q^{\surd}(\gamma_c(\mathcal{C})(2E_b/N_0)), \tag{9.5}$$

where the error coefficient is $K_b(\mathcal{C}) = K_{\min}(\mathcal{C})/k$.

Example 1 (cont.) The nominal coding gain of the Example 1 code is $\gamma_c(\mathcal{C}) = 5/2$ (4 dB) and its error coefficient is $K_b(\mathcal{C}) = K_c(\mathcal{C}) = 1$, so the UBE is

$$P_b(E) \approx Q^{\surd}(5E_b/N_0).$$

Compare the $(8, 4, 4)$ RM block code, which has the same rate (and the same trellis complexity), but has nominal coding gain $\gamma_c(\mathcal{C}) = 2$ (3 dB) and error coefficient $K_b(\mathcal{C}) = 14/4 = 3.5$. □

In general, for the same rate and "complexity" (measured by minimal trellis complexity), convolutional codes have better coding gains than block codes, and much lower error coefficients. They therefore usually yield a better performance *vs.* complexity tradeoff than block codes, when trellis-based ML decoding algorithms such as the VA are used for each. Convolutional codes are also more naturally suited to continuous sequential data transmission than block codes. Even when the application calls for block transmission, terminated convolutional codes usually outperform binary block codes. On the other hand, if there is a decoding delay constraint, then block codes will usually slightly outperform convolutional codes.

Tables 1-3 give the parameters of the best known convolutional codes of short to moderate constraint lengths for rates 1/2, 1/3 and 1/4, which are well suited to the power-limited regime.

We see that it is possible to achieve nominal coding gains of 6 dB with as few as 16 states with rate-1/3 or rate-1/4 codes, and effective coding gains of 6 dB with as few as 32 states, to the accuracy of the union bound estimate (which becomes somewhat optimistic as these codes become more complex). (There also exists a *time-varying* rate-1/2 16-state convolutional code with $d_{\text{free}} = 8$ and thus $\gamma_c = 4$ (6 dB).) With 128 or 256 states, one can achieve effective coding gains of the order of 7 dB. ML sequence detection using the Viterbi algorithm is not a big thing for any of these codes.

Table 9.1: Rate-1/2 binary linear convolutional codes

ν	d_{free}	γ_{c}	dB	K_b	γ_{eff}(dB)
1	3	1.5	1.8	1	1.8
2	5	2.5	4.0	1	4.0
3	6	3	4.8	2	4.6
4	7	3.5	5.2	4	4.8
5	8	4	6.0	5	5.6
6	10	5	7.0	46	5.9
6	9	4.5	6.5	4	6.1
7	10	5	7.0	6	6.7
8	12	6	7.8	10	7.1

Table 9.2: Rate-1/3 binary linear convolutional codes

ν	d_{free}	γ_{c}	dB	K_b	γ_{eff}(dB)
1	5	1.67	2.2	1	2.2
2	8	2.67	4.3	3	4.0
3	10	3.33	5.2	6	4.7
4	12	4	6.0	12	5.3
5	13	4.33	6.4	1	6.4
6	15	5	7.0	11	6.3
7	16	5.33	7.3	1	7.3
8	18	6	7.8	5	7.4

Table 9.3: Rate-1/4 binary linear convolutional codes

ν	d_{free}	γ_{c}	dB	K_b	γ_{eff}(dB)
1	7	1.75	2.4	1	2.4
2	10	2.5	4.0	2	3.8
3	13	3.25	5.1	4	4.7
4	16	4	6.0	8	5.6
5	18	4.5	6.5	6	6.0
6	20	5	7.0	37	6.0
7	22	5.5	7.4	2	7.2
8	24	6	7.8	2	7.6

9.6 More powerful codes and decoding algorithms

We conclude that by use of the optimal VA with moderate-complexity convolutional codes like those in Tables 1-3, we can close about 7 dB of the 12 dB gap between the Shannon limit and uncoded performance at error rates of the order of $P_b(E) \approx 10^{-6}$.

To get closer to the Shannon limit, one uses much longer and more powerful codes with suboptimal decoding algorithms. For instance:

1. *Concatenation* of an inner code with ML decoding with an outer Reed-Solomon (RS) code with algebraic decoding is a powerful approach. For the inner code, it is generally better to use a convolutional code with VA decoding than a block code.

2. A convolutional code with an arbitrarily long constraint length may be decoded by a recursive tree-search technique called *sequential decoding*. There exist various sequential decoding algorithms, of which the fastest is probably the *Fano algorithm*. In general, sequential algorithms follow the best code path through the code trellis (which becomes a tree for long constraint lengths) as long as the path metric exceeds its expected value for the correct path. When a wrong branch is taken, the path begins to look bad; the algorithm then backtracks and tries alternative paths until it again finds a good one.

The amount of decoding computation is a random variable with a Pareto (power-law) distribution; *i.e.,* $\Pr(\text{number of computations} \geq N) \simeq N^{-\alpha}$. A Pareto distribution has a finite mean when the Pareto exponent α is greater than 1. The rate at which $\alpha = 1$ is called the computational cut-off rate R_0. On the AWGN channel, at low code rates, the computational cut-off rate occurs when E_b/N_0 is about 3 dB away from the Shannon limit; thus an effective coding gain of the order of 9 dB can be obtained at error rates of the order of $P_b(E) \approx 10^{-6}$.

3. In the past decade, these techniques have been superseded by capacity-approaching codes such as *turbo codes* and *low-density parity-check codes* with iterative decoding. Such powerful coding schemes will be the principal topic of the latter part of this course.