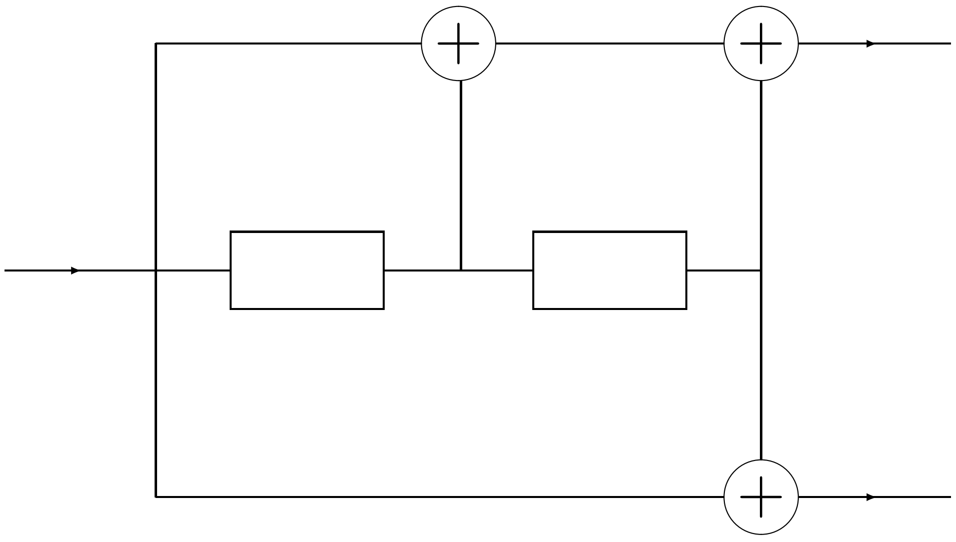
Convolutional Codes

Jyrki Lahtonen



About these notes

The material in these notes was selected to a short introductory course given at the University of Turku in the spring of 1998. The main source is the tutorial chapter "The algebraic theory of convolutional codes" by R.J. McEliece that will appear in *The Handbook of Coding Theory*. His terminology differs from certain earlier practices. As it is expected that his choices will gain acceptance in the years to come, I have adopted all of McEliece's suggested changes and additions. I have also benefited from the book "Introduction to Convolutional Codes with Applications" by A. Dholakia.

The first two chapters follow McEliece's presentation very closely. The third chapter is based on my understanding of some material from Dholakia's book. The term *simple codeword* is due to myself (for lack of a better term), without it I could not have communicated the ideas from Dholakia's book. A related concept of a *fundamental path* through the modified state diagram has appeared in a book by H. van Tilborg.

The last chapter is again an assorted collection of material from McEliece and Dholakia. I chose to present the algebra using subfields of the field of formal Laurent series. I think that this makes it more apparent that blocking a convolutional code does not change the code at all.

For the reader interested in knowing more about the convolutional codes I warmly also recommend the book "Fundamentals of Convolutional Coding" by R. Johannesson and K. Zigangirov.

Introduction

In error correcting coding the message to be sent is usually grouped into shorter sections $\mathbf{u}(0), \mathbf{u}(1), \ldots$ of k bits each. Redundancy is then added such that these k-bit inputs are mapped injectively into the set of n-tuples of bits, where $n > k$. Thus we arrive at a sequence $\mathbf{x}(0), \mathbf{x}(1), \ldots$ of output vectors that are then transmitted into the channel. There are two main schools on what kind of operations we should do to transform the input (the vectors $\mathbf{u}(i)$) into the output (the vectors $\mathbf{x}(i)$).

One of Shannon's theorems loosely speaking states that, if the resulting set of all possible output streams is a "random" subset of the set of all possible sequences of bits, then we can achieve reliable communication provided that the channel is not too bad. In the familiar *block coding* (the topic of the course "Coding Theory") the ith output $\mathbf{x}(i)$ only depends on the input $\mathbf{u}(i)$. E.g. when using a linear code with generator matrix G, the encoding equation

$$\mathbf{x}(i) = \mathbf{u}(i)G$$

holds for all integers i. In order to achieve the required level of randomness it is then necessary that both k and n are relatively large. While not detrimental, this is clearly undesirable in some cases.

In *convolutional coding* (the topic of this course) "randomness" is sought in a different manner: instead of $\mathbf{x}(i)$ being a function of only $\mathbf{u}(i)$ we will allow dependence on the previous inputs $\mathbf{u}(i-1), \mathbf{u}(i-2), \ldots$ as well. Here the parameter i can be thought of as marking the passing of time: in block coding the order in which the k-bit blocks are processed is immaterial and is thus ignored, in convolutional coding the order is important and we keep track of the order of the input blocks with the index i. As we see from the following popular example, in convolutional coding the parameters n and k are much smaller than in block coding.

Let us study the following system, where $k = 1$, $n = 2$ and the input bits $u(i)$ are transformed into the output vectors $\mathbf{x}(i) = (x_1(i), x_2(i))$ by the encoding formulas

$$\begin{cases} x_1(i) &= u(i) + u(i-2), \\ x_2(i) &= u(i) + u(i-1) + u(i-2). \end{cases}$$

Thus the sequence 1 1 0 1 0 0 of input bits is transformed into the sequence 11 10 10 00 01 11 of output pairs of bits. Here we adopted the convention that the inputs prior to the beginning of the transmission are assumed to be zero.

We see that the above encoder at each moment of time i needs to remember the two previous input bits $u(i-1)$ and $u(i-2)$. As the contents of the two memory bits specify the encoding function completely, it is natural to say that at time i the encoder is in the *state* described by the *state vector* $\mathbf{s}(i) = (s_1(i), s_2(i)) = (u(i-1), u(i-2))$. Using the concept of the state vector the above encoding equations can be rewritten as

$$\mathbf{x}(i) = \mathbf{s}(i)\mathcal{C} + \mathbf{u}(i)\mathcal{D},$$

where $\mathcal{C}, \mathcal{D}$ are the matrices

$$\mathcal{C} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \qquad \mathcal{D} = \begin{pmatrix} 1 & 1 \end{pmatrix}.$$

Furthermore we see that the time evolution of the encoder (i.e. how the state of the encoder changes, as time passes by) is completely specified by the requirement that initially the encoder was at the zero state and that

$$\mathbf{s}(i+1) = \mathbf{s}(i)\mathcal{A} + u(i)\mathcal{B},$$

where in this example $\mathcal{A}$ and $\mathcal{B}$ are the matrices

$$\mathcal{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \mathcal{B} = \begin{pmatrix} 1 & 0 \end{pmatrix}.$$

We generalize this situation to arrive at the following definition. Throughout this course we only consider binary codes and let $\mathbf{F}$ denote the field of two elements.

Definition 0.1 An (n, k, m) *convolutional encoder* is the linear system determined by matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}$ and $\mathcal{D}$ with entries in the field $\mathbf{F}$ and of respective types $m \times m$, $k \times m$, $m \times n$ and $k \times n$. The encoder transforms a sequence $\mathbf{u}(i), i = 0, 1, \ldots$ of information words $\in \mathbf{F}^k$ into a sequence $\mathbf{x}(i), i = 0, 1, \ldots$ of codewords $\in \mathbf{F}^n$ with the aid of a sequence of *state vectors* $\mathbf{s}(i), i = 0, 1, \ldots$ belonging to the *state space* $\mathbf{F}^m$ as follows: The system is initialized by $\mathbf{s}(0) = 0$ and for $i \geq 0$ the time-evolution of the system and the codewords are determined by the equations

$$\begin{aligned} \mathbf{s}(i+1) &= \mathbf{s}(i)\mathcal{A} + \mathbf{u}(i)\mathcal{B} \qquad (1) \\ \mathbf{x}(i) &= \mathbf{s}(i)\mathcal{C} + \mathbf{u}(i)\mathcal{D} \qquad (2) \end{aligned}$$

The associated (n, k, m) *convolutional code* is the collection of all the possible output sequences of the encoder. The code has *rate* k/n and *degree* m.

Remark 0.1 A convolutional encoder may be viewed as an automaton. This has motivated some of the terminology.

Remark 0.2 There is an analogous situation in cryptography, where a similar division is made between the block ciphers and the stream ciphers.

Remark 0.3 Strictly speaking the degree is a property of a convolutional encoder rather than a property of the code. Indeed, later on we will develop a method of finding encoders of smallest possible degree. Some sources use different terminology and what we call degree is also often called *memory* or *constraint length.*

Exercises

0.1 *When an (n, k, m) convolutional encoder (determined by matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ of compatible types) is used to encode a sequence $\mathbf{u}(0), \mathbf{u}(1), \ldots, \mathbf{u}(r)$ of input vectors, the transmission is usually extended beyond time r to ensure that the system has reached the zero state again (the reason for this will become evident when we study Viterbi decoding). This is done by imagining that auxiliary zero vectors $\mathbf{u}(i) = 0, i > r$ are also being sent. These extra input vectors are called the terminating zeros. Show that, if the $m \times m$ matrix $\mathcal{A}$ is nilpotent, then a finite number of terminating zeros suffices. Give an example of a $(1, 1, 1)$ encodes, where no finite number of terminating zeros will necessarily put the encoder back into the zero state.*

0.2 *Sharpen the result of the previous exercise by showing that in the case of a nilpotent $m \times m$ matrix $\mathcal{A}$ at most m terminating zeros are needed.*

Chapter 1

Algebraic presentation of convolutional codes

We say that two convolutional encoders are equivalent, if they generate the same code. For reasons that will become apparent later (e.g. when decoding convolutional codes) it is desirable to choose among equivalent encoders the one with the lowest degree. At the moment we have no machinery to study the equivalence of two given encoders. To that end we need such a presentation of a convolutional code, where the state space doesn't appear directly. We first need to review some concepts from algebra.

1.1 Rings of polynomials and power series

Throughout this course D will be an indeterminate. It can be thought of as the operator delaying a given signal by one time unit. Given D we have the polynomial ring

$$\mathbf{F}[D] = \left\{ \sum_{i=0}^{n} a_i D^i \mid n \in \mathbf{N}_0, a_i \in \mathbf{F} \right\}$$

and its field of quotients, the field of rational functions

$$\mathbf{F}(D) = \left\{ \frac{P(D)}{Q(D)} \mid P(D), Q(D) \in \mathbf{F}[D], Q(D) \neq 0 \right\}$$

both with the usual algebraic operations.

We recall that the usual division of polynomials gives the ring $\mathbf{F}[D]$ a structure of a Euclidean domain. We also recall the concept of Smith normal form (defined for matrices over any Euclidean domain) from our Algebra course:

Theorem 1.1 *Let A be an $m \times n$-matrix with entries in $\mathbf{F}[D]$. Then there exist invertible matrices P and Q such that the matrix product*

$$PAQ = \begin{pmatrix} d_1(D) & 0 & \cdots & 0 \\ 0 & d_2(D) & \cdots & 0 \\ 0 & 0 & \ddots & \cdots \end{pmatrix}$$

is diagonal and that the diagonal entries satisfy $d_i(D) \mid d_{i+1}(D)$ for all i.

The polynomials $d_i(D)$ appearing on the diagonal are called the *invariant factors* of the matrix A. We further recall that the matrices P and Q as well as the invariant factors can be found by performing a sequence of elementary row and column operations to the matrix A. We shall need another characterization of the invariant factors.

Lemma 1.1 *Let A be a $k \times n$ matrix with entries in the ring $\mathbf{F}[D]$. Let B be another such matrix gotten from A by an elementary row or column operation (where only multipliers in the ring $\mathbf{F}[D]$ are allowed). Then, for all $r = 1, 2, \ldots, \min\{k, n\}$, the $r \times r$ minors of B are $\mathbf{F}[D]$-linear combinations of the $r \times r$ minors of A.*

Proof. It is enough to check the case of a row operation, for a column operation may viewed as a row operation on the transpose of A. If the row operation is an exchange of two rows, the claim clearly holds. As 1 is the only unit of the ring $\mathbf{F}[D]$, there are no elementary row operations of the scalar multiplication type. If the row operation consists of adding the entries of a source row (multiplied by a polynomial $p(x)$) to the corresponding entries in another target row, then there are several possibilities: Any such $r \times r$-minors of B that don't include the target row are obviously minors of A as well. Such $r \times r$-minors of B that include both the target and the source rows are also minors of A — effectively we are then performing a determinant preserving elementary row operation on a submatrix of A. The most interesting cases are such $r \times r$-minors that include the target row, but don't include the source row. However, we may express such a minor as a sum of two $r \times r$-determinants by splitting the target row to the sum of the original target row and the source row times $p(x)$. Here the first summand is again a minor of A also. We can factor out $p(x)$ from the target row of the second summand. The other factor is a minor of A with rows possibly in the wrong order. The claim then follows in this case as well. ∎

Corollary 1.1 *Let A, P, Q be as in Theorem 1.1. Then for all applicable ℓ the product $d_1(D)d_2(D) \cdots d_\ell(D)$ is the greatest common divisor of all the $\ell \times \ell$-minors (=subdeterminants) of the matrix A. In particular, the invariant factors are uniquely determined.*

Proof. From Lemma 1.1 it actually follows that, if a matrix B is gotten from another matrix A by an elementary row or a column operation, then for all r the greatest common divisors of the $r \times r$ minors of A and B are equal. This is because a row or a column operation can always be reversed.

Thus a sequence of elementary row and column operations always preserves the greatest common divisors of minors of a prescribed size r. However, for a matrix in the normal form, this greatest common divisor is clearly the product of the r first invariant factors.

The last claim now follows from the facts that a greatest common divisor is unique up to a unit factor and that 1 is the only unit of $\mathbf{F}[D]$. ∎

Square matrices with polynomial entries that have an inverse with polynomial entries are also called *unimodular*. It follows from Cramer's rule that the determinant of such a matrix M must be a unit in the polynomial ring, i.e. equal to one. As the determinant of a square matrix is the product of its invariant factors, this happens if and only if all the invariant factors are also equal to 1. Thus we have shown that a square matrix is unimodular, iff it can be reduced to the identity matrix with elementary row operations (where we only allow polynomial multipliers), or iff its invariant factors are all equal to 1. Equivalently, a unimodular matrix is a product of elementary polynomial matrices of the same size.

Example 1.1 Find the invariant factors of the matrices

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & D & D^2 \\ 1 & D^2 & D^4 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}$$

We shall also need the ring of formal power series in the unknown D is (formal = not interested in convergence properties)

$$\mathbf{F}[[D]] = \left\{ \sum_{i \geq 0} a_i D^i \mid a_i \in \mathbf{F} \right\}.$$

Again the usual 'polynomial-like' operations make this into an integral domain. Unlike the ring $\mathbf{F}[D]$ of 'finite' power series this ring has many units.

Proposition 1.1 *The power series $a(D) = a_0 + a_1 D + a_2 D^2 + \cdots \in \mathbf{F}[[D]]$ is a unit, iff $a_0 \neq 0$ (i.e., $a_0 = 1$).*

Proof. An inverse of such a power series can be explicitly constructed by recursively solving the unknown coefficients of the inverse series. ∎

Example 1.2 Find the inverse elements of $1 + D$ and $(1 + D)^3$.

Corollary 1.2 *All the elements $a(D) \in \mathbf{F}[[D]] \setminus \{0\}$ can be uniquely expressed in the form*

$$a(D) = D^m u(D),$$

where $u(D)$ is a unit of $\mathbf{F}[[D]]$ and m is a non-negative integer.

We will further need the ring of formal Laurent series

$$\mathbf{F}((D)) = \left\{ \sum_{i \geq m} a_i D^i \mid a_i \in \mathbf{F}, m \in \mathbf{Z} \right\}.$$

This is again a ring with respect to the usual definitions. However, observe that we cannot allow such power series that have an infinite number of terms with negative exponents: without any convergence theory the product of two such 'Laurent series' would not be well defined. Actually it turns out that $\mathbf{F}((D))$ is a field and not just any field:

Corollary 1.3 *$\mathbf{F}((D))$ is the field of quotients of the domain $\mathbf{F}[[D]]$.*

Proof. Another exercise. ∎

We have the following inclusions among these rings/fields:

$$\mathbf{F}[D] \subset \mathbf{F}[[D]], \mathbf{F}[D] \subset \mathbf{F}(D), \mathbf{F}(D) \subset \mathbf{F}((D)), \mathbf{F}[[D]] \subset \mathbf{F}((D)).$$

Elements of the ring $\mathbf{F}[[D]]$ is are also called *causal Laurent series*, those of $\mathbf{F}(D)$ *realizable Laurent series* and elements of the intersection (inside the field $\mathbf{F}((D))$)

$$\mathbf{F}[[D]] \cap \mathbf{F}(D) = \left\{ \frac{P(D)}{Q(D)} \mid P(D), Q(D) \in \mathbf{F}[D], Q(0) \neq 0 \right\}$$

are called *causal rational functions*.

The number (zero, natural number or infinity) of non-zero terms in a Laurent series is called its *weight*. Thus every polynomial is of finite weight and every Laurent series of a finite weight can be written in the form $D^m P(D)$, where $P(D)$ is a polynomial. Also for a Laurent series $a(D)$ of a finite weight we can define its degree to be the exponent of its highest degree term.

In what follows we will also encounter power series and polynomials with vector coefficients. These should be thought of as elements of a suitable cartesian power $\mathbf{F}((D))^n$ that we will denote by $\mathbf{F}^n((D))$ (similarly for other rings and fields). This is a vector space over the field $\mathbf{F}((D))$ (and a module over the subrings $\mathbf{F}[D]$ and $\mathbf{F}[[D]]$). So we mean e.g.

$$(1,0,1) + (0,1,1)D + (1,0,0)D^2 = (1+D^2, D, 1+D)$$

and make the similar identifications with matrices having polynomials or power series as entries.

We can then extend the concept of weight to power series with vector coefficients and declare that the weight of a vector (when finite) is the sum of the weights of the components. With the above identification this leads to a rather natural concept of a weight.

1.2 Convolutional codes and power series

We are now ready to redefine the concept of a convolutional code. The idea is simply to use power series as generating functions of a sequence of vectors. Thus we present a sequence of input vectors $\mathbf{u}(i) \in \mathbf{F}^k, i = m, m+1, \ldots$ as the series

$$\mathbf{u}(D) = \sum_{i \geq m} \mathbf{u}(i) D^i \in \mathbf{F}^k((D)).$$

Here we allow the transmission to begin at any time (not just at $i = 0$), but we do require the system to start at some time $m \in \mathbf{Z}$. This is a very natural relaxation and from now on we have the liberty of choosing the origin of our time axis. Similarly we define

$$\mathbf{x}(D) = \sum_{i \geq m} \mathbf{x}(i) D^i \in \mathbf{F}^n((D)) \qquad \text{and} \qquad \mathbf{s}(D) = \sum_{i \geq m} \mathbf{s}(i) D^i \in \mathbf{F}^m((D)).$$

Definition 1.1 An (n, k) *convolutional code* is such a k-dimensional subspace (over the field $\mathbf{F}((D))$) of the space $\mathbf{F}^n((D))$ that it has a basis consisting of vectors belonging to $\mathbf{F}^n(D)$.

Observe that changing the origin of the time axis from zero to m simply amounts to multiplication by D^m. The use of Laurent series thus allows us to go back and forth in time as need may be.

We have the obvious task of showing that this is equivalent to our earlier definition. We show that the earlier definition will lead to a convolutional code in the sense of this new definition. The other direction will become apparent, when we construct encoders for convolutional codes in the sense of this latter definition.

Let us begin with an (n, k, m) convolutional encoder determined by matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}$ and $\mathcal{D}$. Multiplying the defining equations (1) and (2) by D^i and summing over i we arrive at the following pair of matrix equations

$$\left\{ \begin{array}{rcl} \mathbf{x}(D) & = & \mathbf{s}(D)\mathcal{C} + \mathbf{u}(D)\mathcal{D} \\ D^{-1}\mathbf{s}(D) & = & \mathbf{s}(D)\mathcal{A} + \mathbf{u}(D)\mathcal{B} \end{array} \right.$$

The Cramer's formula for the elements of the inverse matrix shows that the matrix $D^{-1}I_m + \mathcal{A}$ has an inverse in the ring of $m \times m$-matrices over the field $\mathbf{F}(D)$. Hence we can solve $\mathbf{s}(D)$ from the latter equation and substitute the result to the former equation and get the equation $\mathbf{x}(D) = \mathbf{u}(D)G$, where

$$G(D) = \mathcal{D} + \mathcal{B}\left(D^{-1}I_m + \mathcal{A}\right)^{-1}\mathcal{C}$$

clearly has entries in the field $\mathbf{F}(D)$. Obviously the rows of the $k \times n$ matrix G form a basis of the required type.

Example 1.3 Find a polynomial generator matrix for the example convolutional code of the introduction.

As we can always multiply basis vectors with non-zero scalars to get another basis, we can always arrange the generator matrix G of a convolutional code to have polynomial entries — simply clear the denominators by multiplying each row of G with the least common multiple of of the denominators in that row. Hence a convolutional code always has non-zero elements of finite weight. This motivates the following definition.

Definition 1.2 The *free distance* of a convolutional code is the minimum weight of a non-zero element of the code.

The free distance takes the role of the minimum distance of a block code — the higher the free distance the better the error correcting capability of the code.

Example 1.4 Show that the free distance of the convolutional code in example 1.3 is 5.

1.3 Polynomial generator matrices

A generator matrix $G(D)$ of an (n, k) convolutional code C is called a polynomial generator matrix of C, if all of its entries are polynomials in $\mathbf{F}[D]$. A convolutional code has many polynomial generator matrices.

Example 1.5 Show that all the following matrices generate the same $(4, 2)$ convolutional code

$$\begin{aligned}
G_1 &= \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}, \\
G_2 &= \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 1+D & 0 & 1 & D \end{pmatrix}, \\
G_3 &= \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{pmatrix}, \\
G_4 &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{pmatrix}.
\end{aligned}$$

We shall later see (crudely speaking) that the more delay operators there are in the generator matrix the more complicated both encoding and decoding will become. Thus generator matrices involving lower degree polynomials are to be preferred. From the above example we saw that row operations involving multipliers from $\mathbf{F}(D)$ rather than $\mathbf{F}[D]$ may further simplify the generator matrix. The obvious questions are: 1) How do we measure the complexity of a generator matrix? 2) How can we know that the generator matrix is in a simplest possible form? We will study these two questions in this section.

Let $G(D) = (g_{ij}(D))$ be a $k \times n$ polynomial matrix. We will denote the ith row G by $\mathbf{g}_i(D) = (g_{i1}(D), \ldots, g_{in}(D)) \in \mathbf{F}^n[D]$, and we define the degree of $\mathbf{g}_i(D)$ as the maximum degree of its components. We then define the *internal degree* and *external degree* of $G(D)$ as follows.

$$\begin{aligned} \operatorname{intdeg} G(D) &= \text{maximum degree of } G(D)\text{'s } k \times k \text{ minors} \\ \operatorname{extdeg} G(D) &= \text{sum of the row degrees of } G(D). \end{aligned}$$

The following two definitions will be essential in our study of polynomial generator matrices.

Definition 1.3 A $k \times n$ polynomial matrix $G(D)$ is called *basic*, if it has the minimum possible internal degree among the polynomial matrices of the form $T(D)G(D)$, where $T(D)$ is a non-singular $k \times k$ matrix over $\mathbf{F}(D)$

Definition 1.4 A $k \times n$ polynomial matrix $G(D)$ is called *reduced*, if it has the minimum possible external degree among the matrices of the form $T(D)G(D)$, where $T(D)$ is unimodular.

As unimodular matrices are obtained from the identity matrix by elementary row operations (with coefficients in $\mathbf{F}[D]$) an equivalent definition would be to declare $G(D)$ to be reduced, if its external degree cannot decrease in a sequence of elementary row operations.

We first prove the following basic fact.

Lemma 1.2 *Let $G(D)$ be a $k \times n$ polynomial matrix.*

(A) If $T(D)$ is any nonsingular polynomial $k \times k$ matrix (i.e. $\det T(D) \neq 0$), then

$$\operatorname{intdeg} T(D)G(D) = \operatorname{intdeg} G(D) + \deg \det T(D).$$

In particular $\operatorname{intdeg} T(D)G(D) \geq \operatorname{intdeg} G(D)$ *with equality, iff $T(D)$ is unimodular.*

(B)

$$\operatorname{intdeg} G(D) \leq \operatorname{extdeg} G(D).$$

Proof. (a) The $k \times k$ submatrices of $T(D)G(D)$ are simply the $k \times k$ submatrices of $G(D)$ multiplied by $T(D)$. Thus the $k \times k$ minors of $T(D)G(D)$ are simply the $k \times k$ minors of $G(D)$ multiplied by $\det T(D)$. The claim follows.

(b) Let e_i be the degree of the row $\mathbf{g}_i(D)$, so each entry from the ith row will have degree $\leq e_i$. Every term in the expansion of any $k \times k$ minor is then of degree at most $e_1 + e_2 + \cdots + e_k = \operatorname{extdeg} G(D)$. ∎

We next prove two important theorems that list several useful properties of basic and reduced polynomial matrices.

Theorem 1.2 *A $k \times n$ polynomial matrix $G(D)$ is basic, iff any one of the following six conditions is satisfied.*

(1) The invariant factors $d_1(D), d_2(D), \ldots, d_k(D)$ of $G(D)$ are all equal to 1.

(2) The greatest common divisor of the $k \times k$ minors of $G(D)$ is 1.

(3) The matrix $G(\alpha)$ has rank k for any α that is algebraic over the field $\mathbf{F}$.

(4) $G(D)$ has a polynomial right inverse, i.e. there exists an $n \times k$ matrix $H(D)$ such that $G(D)H(D) = I_k$.

(5) If $\mathbf{x}(D) = \mathbf{u}(D)G(D)$ and $\mathbf{x}(D) \in \mathbf{F}^n[D]$, then $\mathbf{u}(D) \in \mathbf{F}^k[D]$. ("Polynomial output implies polynomial input.")

(6) $G(D)$ can be completed to a unimodular matrix by adding $n-k$ suitable polynomial rows.

Proof. We shall prove the following implications: Basic $\Rightarrow$(1) $\Rightarrow$(2) $\Rightarrow$(4) $\Rightarrow$(5) $\Rightarrow$Basic; (2)$\Leftrightarrow$(3); (1)$\Leftrightarrow$(6).

We have already seen (lectures/exercises) that the greatest common divisor of the $k \times k$ minors is the product of the invariant factors. Thus obviously (1) $\Leftrightarrow$(2).

• Basic $\Rightarrow$(1): Let P and Q be the unimodular matrices that take $G(D)$ to its normal form $\Gamma(D)$, i.e.

$$PG(D)Q = \Gamma(D) = \begin{pmatrix} d_1(D) & 0 & 0 & \cdots & 0 \\ 0 & d_2(D) & 0 & \cdots & 0 \\ 0 & \cdots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & d_k(D) & \cdots \end{pmatrix}.$$

So $G(D)$ and $G'(D) = \Gamma(D)Q^{-1} = PG(D)$ generate the same code and $\operatorname{intdeg} G(D) = \operatorname{intdeg} G'(D)$ as P and Q are unimodular. If there are non-constant entries in $\Gamma(D)$, then we can form another *polynomial* generator matrix

$$G''(D) = \operatorname{diag}\left(d_1(D)^{-1}, \dots, d_k(D)^{-1}\right) G'(D) = \left(I_k \mid 0_{n-k\times k}\right) Q^{-1}$$

that clearly has internal degree = $\operatorname{intdeg} G(D) - \sum_{i=1}^{k} \deg d_i(D)$ contradicting our hypothesis that $G(D)$ is basic.

• (2) $\Rightarrow$(4): Let us fix k columns of $G(D)$. There are $\binom{n}{k}$ ways to do this, we index them by a parameter $\nu = 1, 2, \dots, \binom{n}{k}$. By Cramer's rule the cofactors of this resulting submatrix $G_\nu(D)$ form a matrix $C_\nu(D)$ with the property $G_\nu(D)C_\nu(D) = p_\nu(D)I_k$, where $p_\nu(D)$ is the determinant of $G_\nu(D)$. We can then add $n-k$ rows of all zeros to the matrix $C_\nu(D)$ to form an $n \times k$ matrix H_ν such that the non-zero rows pair with the chosen k columns of $G(D)$. Thus $G(D)H_\nu = p_\nu(D)I_k$. As the greatest common divisor of the polynomials $p_\nu(D)$ is equal to one, there exist polynomials $q_\nu(D), \nu = 1, \dots, \binom{n}{k}$ such that

$$\sum_{\nu=1}^{\binom{n}{k}} q_\nu(D)p_\nu(D) = 1.$$

The polynomial matrix

$$H(D) = \sum_{\nu=1}^{\binom{n}{k}} q_\nu(D)H_\nu$$

then clearly is a sought right inverse to $G(D)$.

• (4) $\Rightarrow$(5): Now $\mathbf{u}(D) = \mathbf{u}(D)G(D)H(D) = \mathbf{x}(D)H(D)$, so this is trivial.

• (5) $\Rightarrow$Basic: If $T(D)G(D)$ is a polynomial matrix then by applying (5) to the rows of this matrix product we see that, in fact, $T(D)$ must be a polynomial matrix, too. Our Lemma 1.2 then tells us that $\operatorname{intdeg} T(D)G(D) \geq \operatorname{intdeg} G(D)$ and thus $G(D)$ is basic.
• (2) $\Leftrightarrow$(3): Assume (2). Let α be an arbitrary element in the algebraic closure of $\mathbf{F}$, let $p(D) \in \mathbf{F}[D]$ be the minimal polynomial of α and let $q_\nu(D), \nu = 1, \ldots \binom{n}{k}$ be the $k \times k$ minors of $G(D)$. Then $G(\alpha)$ is of rank less than k if and only if $q_\nu(\alpha) = 0$ for all ν, i.e. $p(D)$ divides all the polynomials $q_\nu(D)$. This contradicts our assumption (2). Conversely, if the polynomials $q_\nu(D)$ have a non-trivial common irreducible divisor $p(D)$, then $p(D)$ has a root α in some algebraic extension of $\mathbf{F}$. All the $k \times k$ minors of the matrix $G(\alpha)$ then vanish, hence $G(\alpha)$ has rank less than k.
• (1) $\Leftrightarrow$(6): Assume (1). Then the Smith normal form equation for G can be written as

$$G = P^{-1} \begin{pmatrix} I_k & 0_{k,n-k} \end{pmatrix} Q^{-1}.$$

Write $Q^{-1} = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$, where B_1 contains the k first rows of Q^{-1} and B_2 the remaining $n-k$ rows. Then $G = P^{-1}B_1$ and the matrix $\begin{pmatrix} P^{-1}B_1 \\ B_2 \end{pmatrix}$ is unimodular as it is obtained from Q^{-1} by applying elementary row operations on the first k rows. Conversely, if $A = \begin{pmatrix} G(D) \\ H(D) \end{pmatrix}$ is unimodular, then the Smith normal form equation

$$G(D) = I_k \begin{pmatrix} I_k & 0_{k,n-k} \end{pmatrix} A$$

shows that the invariant factors of $G(D)$ are all 1. ∎

Theorem 1.3 *A $k \times n$ polynomial matrix $G(D)$ is reduced if and only if one of the following three conditions is satisfied.*
(1) If we define the "indicator matrix for the highest-degree terms in each row" $\bar{G}$ by $\bar{G}_{ij} =$ the coefficient of D^{e_i} in $g_{ij}(D)$, where e_i is the degree of $\mathbf{g}_i(D)$, then $\bar{G}$ has full rank k.
(2) $\operatorname{extdeg} G(D) = \operatorname{intdeg} G(D)$.
(3) "The predictable degree property": For any $\mathbf{u}(D) \in \mathbf{F}^k[D]$

$$\deg\left(\mathbf{u}(D)G(D)\right) = \max_{1 \leq i \leq k} \left(\deg u_i(D) + \deg \mathbf{g}_i(D)\right).$$

Proof. We shall prove that Reduced $\Rightarrow$(1) $\Rightarrow$(2) $\Rightarrow$Reduced and that (1) $\Leftrightarrow$(3). Let again $\mathbf{g}_i$ be the ith row of $G(D)$ and e_i its degree.
• Reduced $\Rightarrow$(1): Suppose (1) is false. Then there is a linear dependency relation among the rows of $\bar{G}$ and we can find a non-zero vector $\mathbf{a} = (a_1, a_2, \ldots, a_k) \in \mathbf{F}^k$ such that $\mathbf{a}\bar{G} = 0$. Without loss of generality we can assume that $e_1 \leq e_2 \leq \cdots \leq e_k$. Let ℓ be the highest index with the property $a_\ell \neq 0$. It follows that the linear combination

$$\mathbf{g}'_\ell = a_1 D^{e_\ell - e_1}\mathbf{g}_1 + a_2 D^{e_\ell - e_2}\mathbf{g}_2 + \cdots + a_\ell D^{e_\ell - e_\ell}\mathbf{g}_\ell$$

is of degree strictly less than e_ℓ. A sequence of elementary row operations then replaces $\mathbf{g}_\ell$ with $\mathbf{g}'_\ell$ contradicting our hypothesis.

• (1) $\Rightarrow$(2): Let $\bar{G}_\nu, \nu = 1, 2, \ldots, \binom{n}{k}$ be the $k \times k$ submatrices of $\bar{G}$. By our assumption there exists ν_0 such that $\det \bar{G}_{\nu_0} \neq 0$. Let us then form the corresponding $k \times k$ submatrix G_{ν_0} of $G(D)$. Then the coefficient of $D^{e_1+e_2+\cdots+e_k}$ in $\det G_{\nu_0}$ is $\det \bar{G}_{\nu_0} \neq 0$. Thus $\operatorname{intdeg} G \geq e_1 + e_2 + \cdots + e_k = \operatorname{extdeg} G$. The reverse inequality holds always by Lemma 1.2.
• (2) $\Rightarrow$(Reduced): Assume that $\operatorname{intdeg} G(D) = \operatorname{extdeg} G(D)$ and that $T(D)$ is an arbitrary $k \times k$ unimodular matrix. Then by Lemma 1.2

$$\operatorname{extdeg} T(D)G(D) \geq \operatorname{intdeg} T(D)G(D) \qquad \text{and} \qquad \operatorname{intdeg} T(D)G(D) = \operatorname{intdeg} G(D).$$

Combining these with our assumption we get $\operatorname{extdeg} T(D)G(D) \geq \operatorname{extdeg} G(D)$, which proves that $G(D)$ is reduced.
• (1) $\Leftrightarrow$(3): Let $\mathbf{u}(D) = (u_1(D), \ldots, u_k(D)) \in \mathbf{F}^k[D]$ and $\mathbf{x}(D) = \mathbf{u}(D)\mathbf{g}(D)$, so

$$\mathbf{x}(D) = u_1(D)\mathbf{g}_1(D) + \cdots + u_k(D)\mathbf{g}_k(D).$$

If the degree of $u_i(D)$ is d_i, we see that the degree of $\mathbf{x}(D)$ is at most $d = \max_i (d_i + e_i)$. This is what we "predict" the degree of $\mathbf{x}(D)$ to be. To test this prediction, we note that the vector of coefficients of D^d in $\mathbf{x}(D)$ is $\mathbf{b} = (a_1, \ldots, a_k)\bar{G} \in \mathbf{F}^n$, where a_i is the coefficient of D^{d-e_i} in $u_i(D)$. But at least one a_i is non-zero, so $\mathbf{b}$ can be the zero vector for some choice of $\mathbf{u}(D)$ if and only if $\bar{G}$ has rank less than k, otherwise the prediction is true for all choices of $\mathbf{u}(D)$. ∎

It follows immediately from our definitions that the basic polynomial generator matrices for a given code have the lowest possible internal degree. However, it turns out that the polynomial generator matrices of the lowest possible external degree are more important

Definition 1.5 A polynomial generator matrix for a convolutional code C is called *canonical*, if it has the smallest possible external degree. This minimum external degree is called the *degree* of the code C.

Later on we will see that this definition of the degree is in accordance with our earlier definition — the minimum external degree for a generator matrix of a convolutional code C turns out to be equal to the smallest possible dimension of the state space of an encoder of C.

The following result shows that canonical polynomial generator matrices have many nice properties.

Theorem 1.4 *A polynomial generator matrix $G(D)$ for the convolutional code C is canonical if and only if it is both basic and reduced.*

Proof. Let us first assume that $G(D)$ is canonical. As the external degree of $G(D)$ obviously cannot be decreased by any sequence of elementary row operations, $G(D)$ is automatically reduced. In order to see that $G(D)$ must also be basic we need to compare its internal degree to that of a suitable basic polynomial generator matrix. So let $G_0(D)$ be such a basic generator matrix of C, for which the external degree is as small as possible. We shall first show that $G_0(D)$ is then necessarily reduced.

To that end let $T(D)$ be a unimodular $k \times k$ matrix. Then the matrix product $T(D)G_0(D)$ is also basic, as by Lemma 1.2 $\operatorname{intdeg} T(D)G_0(D) = \operatorname{intdeg} G_0(D)$ (= the common internal degree of all the basic generator matrices). By the choice of $G_0(D)$ $\text{extdeg}\, T(D)G_0(D) \geq \text{extdeg}\, G_0(D)$. This implies that $G_0(D)$ is, indeed, reduced.

Lemma 1.2 gives again that $\text{intdeg}\, G(D) \leq \text{extdeg}\, G(D)$ so we have the following chain of inequalities

$$\text{intdeg}\, G_0(D) \leq \text{intdeg}\, G(D) \leq \text{extdeg}\, G(D) \leq \text{extdeg}\, G_0(D). \tag{1.1}$$

But we just saw that $G_0(D)$ is reduced, so by Theorem 1.3 $\text{intdeg}\, G_0(D) = \text{extdeg}\, G_0(D)$ and so we have equality throughout the above chain of inequalities. Thus $\text{intdeg}\, G(D) = \text{intdeg}\, G_0(D)$ and $G(D)$ is also basic.

Conversely let us assume that $G(D)$ is basic and reduced. Let $G_0(D)$ be any other polynomial generator matrix for C. By Lemma 1.2 $\text{extdeg}\, G_0(D) \geq \text{intdeg}\, G_0(D)$. Since $G(D)$ is basic we get $\text{intdeg}\, G_0(D) \geq \text{intdeg}\, G(D)$ and since $G(D)$ is reduced we also have $\text{intdeg}\, G(D) = \text{extdeg}\, G(D)$ by Theorem 1.3. Altogether we get that $\text{extdeg}\, G_0(D) \geq \text{extdeg}\, G(D)$ and thus $G(D)$ is canonical. ∎

The following corollary is an immediate consequence.

Corollary 1.4 *The minimal internal degree of any polynomial generator matrix for a given convolutional code C is equal to the degree of C. This internal degree is shared by all the basic generator matrices of C.*

Not only will the canonical generator matrices have the lowest possible external degree. The next result shows that they have in a sense the lowest possible row degrees as well.

Theorem 1.5 *Let $G(D)$ be a canonical generator matrix for C and let $G'(D)$ be any other polynomial generator matrix. Assume that their rows are ordered such that the respective row degrees $e_1 \leq e_2 \leq \cdots \leq e_k$ and $f_1 \leq f_2 \leq \cdots \leq f_k$ are in ascending order. Then for all i we have $e_i \leq f_i$. In particular all the canonical generator matrices share the same sequence of row degrees.*

Proof. Assume contrariwise that there exists an index j such that $e_i \leq f_i$ for $i = 1, 2, \ldots, j-1$ and $e_j > f_j$. Let $\mathbf{g}_i$ (resp. $\mathbf{g}'_i$) be the rows of the matrix $G(D)$ (resp. $G'(D)$). As $G(D)$ is canonical, hence basic, by the "polynomial output implies polynomial input" property (Theorem 1.2 part (5)) the vectors $\mathbf{g}'_i$ can be expressed as such linear combinations of the vectors $\mathbf{g}_i$ where only polynomial coefficients are needed. By "the predictable degree property" (Theorem 1.3 part (3)) only the vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{j-1}$ can appear in the expressions for $\mathbf{g}'_1, \mathbf{g}'_2, \ldots, \mathbf{g}'_j$. But this implies that the first j rows of $G'(D)$ are linearly dependent over the field $\mathbf{F}(D)$. This is a contradiction.

The latter claim follows immediately as when applying the result to any two canonical generator matrices G_1 and G_2 either one of them can take the role of $G(D)$ in the proven claim. ∎

We shall call the common values of the row degrees e_i of canonical generator matrices the *Forney indices* of the code C and the maximal row degree e_k the *memory* of the code. Some sources give what we call *degree* the name *memory.* Other terms for the minimal external degree are *constraint length* and *state complexity.*

Now that we have a well defined concept of the memory of a convolutional code (i.e. not dependent on the properties of a particular encoder) we can call an (n, k) convolutional code of degree m an (n, k, m) code. Furthermore, if the code in question has free distance d we call it an (n, k, m, d) code.

Example 1.6 Determine, whether the different polynomial generator matrices for the code of example 1.5 are basic, reduced or canonical. Find the Forney indices and memory of the code of example 1.5.

Example 1.7 Show that if G is a polynomial generator matrix for an (n, k) code C and PGQ is its Smith normal form, then the matrix Γ consisting of the k first rows of the polynomial matrix Q^{-1} is a basic generator matrix for C. Show further that repeated applications of the unimodular external degree reducing transformation in the proof of Theorem 1.3 lead to a canonical generator matrix after a finite number of steps.

Example 1.8 Let C be the $(5, 1, 2)$ code generated by the canonical matrix

$$G(D) = (1 + D^2, 1 + D^2, 1 + D + D^2, 1 + D + D^2, 1 + D + D^2).$$

Show that the free distance of $G(D)$ is 13. Show that no $(5, 1, 2, d)$ convolutional codes with $d > 13$ exist. Thus we may say that C is a distance optimal code.

We close this chapter with the remark that although polynomial generator matrices are very natural for the development of the theory, in practice generator matrices whose entries are causal rational functions are also important. This is because it is desirable to use a so called *systematic generator matrix*, i.e. an $n \times k$ generator matrix that has the $k \times k$ identity matrix as a block in the k leftmost columns. For most convolutional codes there are no polynomial systematic generator matrices, so rational matrix entries are then necessary. A generator matrix G for a convolutional code can always be put into systematic form by a sequence of row operations *using multipliers from the field* $\mathbf{F}(D)$ (instead of the ring $\mathbf{F}[D]$). Exchanges of columns may first be necessary to achieve a situation where the first k columns of G are linearly independent over $\mathbf{F}(D)$.

Example 1.9 Show that the matrix

$$G_5 = \begin{pmatrix} 1 & 0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0 & 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix}$$

is a systematic causal generator matrix for the $(4, 2)$ code of example 1.5.

Exercises

1.1 *Show that if the matrix $\mathcal{A} \in \mathcal{M}_{m\times m}(\mathbf{F})$ is nilpotent, then the inverse of the matrix $D^{-1}I_m + \mathcal{A}$ belongs to $\mathcal{M}_{m\times m}(\mathbf{F}[D])$. Hint: In an earlier algebra course you have probably seen a useful formula for the inverse of an element of the form $1 + n$, where n is nilpotent.*

1.2 *Find the coefficients $a_i \in \mathbf{F}$ of the series*

$$\left(1 + D + D^2\right)^{-1} = \sum_{i\geq 0} a_i D^i \in \mathbf{F}[[D]].$$

If you feel like it, you may study the relation of these coefficients to the integers in the Fibonacci sequence.

1.3 *Let C be the convolutional code generated by*

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

Show that the free distance of C is 5.

1.4 *Let C be the convolutional code generated by*

$$G(D) = \begin{pmatrix} 1+D^2 & 1+D+D^2 & 1+D+D^2 & 1+D+D^2 \end{pmatrix}.$$

Show that the free distance of C is 10.

1.5 *Find the internal and external degrees of the matrix*

$$G = \begin{pmatrix} 1 & 0 & 0 & 1+D \\ 0 & 1 & 0 & 1+D^2 \\ 0 & 0 & 1 & 1+D+D^2 \end{pmatrix}.$$

1.6 *Find a polynomial right inverse for the matrix*

$$G_3 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{pmatrix}$$

of Example 1.5. If possible find the input $\mathbf{u}(D)$ generating the output $\mathbf{x}(D) = \mathbf{u}(D)G_3 = (1+D, D^2+D^3, 1+D^3, D+D^2)$?

1.7 *Show that the following matrices generate the same code. Are they basic, reduced or canonical?*

$$G_1 = \begin{pmatrix} 1 & 1+D & D \\ 1+D & 1+D+D^2 & 1+D+D^2 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 1 & 1+D & D \\ 1 & 1 & 1+D \end{pmatrix}$$

1.8 *Find a canonical generator matrix for the $(4,3)$ code C generated by the matrix G of problem 1.5. What are the Forney indices of C?*

1.9 *We say that a generator matrix $G(D)$ of an (n,k) code is* systematic*, if it is of the block form $G(D) = (I_k \mid A)$, where I_k is the identity matrix and A is a polynomial $k \times (n-k)$ matrix, e.g. the matrix of exercise 1.5 is systematic. Show that a systematic generator matrix is always basic, but not necessarily reduced.*

1.10 *Let $G(D)$ be a basic polynomial generator matrix of an (n,k) code. Show that $G(D)$ has the following "finite output implies finite input" property: If $\mathbf{x}(D) = \mathbf{u}(D)G(D)$ has finite weight, then the weight of the input $\mathbf{u}(D)$ is also finite. Give an example of a non-basic polynomial generator matrix that also has this property. A generator matrix that doesn't have this property is called* catastrophic.

1.11 *Assume that $G(D)$ is a canonical generator matrix of an (n,k) convolutional code. Form a $(k-1) \times n$ matrix $G'(D)$ by deleting one of the rows of $G(D)$. Show that $G'(D)$ is necessarily also a canonical polynomial matrix.*

1.12 *Let $G(D) = (1+D+D^4, 1+D^2+D^3+D^4)$. Show that $G(D)$ is a canonical generator matrix for a $(2,1,4)$ convolutional code C and find a polynomial right inverse $H(D)$ for $G(D)$.*

1.13 *Let us transform our favorite* $(2,1)$ *code with generator matrix*

$$G(D) = (1 + D^2, 1 + D + D^2)$$

into a $(4,2)$ *code* C *by "running the clock at half speed" so that at each time* i *two input bits arrive and four output bits are transmitted, i.e. the input vector at time* i *is*

$$\mathbf{v}(i) = (u(2i), u(2i + 1))$$

and the output vector at time i *is*

$$\mathbf{y}(i) = (x_1(2i), x_1(2i + 1), x_2(2i), x_2(2i + 1)),$$

where $u(j)$ *(resp.* $x_i(j)$*) refer to the input (resp. output) bits of the original encoder. Find a canonical generator matrix for the* $(4,2)$ *code* C*. I reveal the fact that the Forney indices of* C *are both* $= 1$.

1.14 *A* $(2,1)$*-code is said to have "Easy-Look-In" (ELI)-property, if the input bits* $u(i), i \geq 0$ *can be easily computed in terms of the output bits* $x_1(i), x_2(i), i \geq 0$ *either from the formula*

$$u(i) = x_1(i) + x_1(i - 1) + x_2(i - 1), \tag{1}$$

or from the formula (the roles of the output signals are reversed)

$$u(i) = x_1(i - 1) + x_2(i) + x_2(i - 1). \tag{2}$$

What are the corresponding equations relating the signals $u(D), x_1(D)$ *and* $x_2(D)$*? Find a right inverse to the generator matrix of an ELI-code in the two cases (1) and (2).*

1.15 (An important and easy exercise) *Why is it necessary to require that the entries of the generator matrix must be causal?*

Chapter 2

Encoders for convolutional codes

In this chapter we will study the construction of encoders for convolutional codes. We will also touch the subject of construction of physical encoder circuits such as the one on the cover page. Although closer to engineering, such diagrams aid mathematicians as well in understanding what is going on.

We will describe, how to construct a circuit closely matching a given polynomial generator matrix for a convolutional code. It turns out that the number of delay elements in the circuit is exactly the external degree of the generator matrix (though sometimes one may be able to get away with fewer delays). It comes then as no surprise that the canonical generator matrices will yield the smallest (=best) circuits. We will close this chapter by proving a theorem effectively stating that the degree of a code is, indeed, the minimum number of delays one will need in an encoder. While sometimes other consideration may be important (such as easy recovery of the input bits from the output), this result more or less tells us to concentrate on the canonical generator matrices.

2.1 Physical encoders

Let us take a closer look what happens, when we use a polynomial generator matrix $G(D) = (g_{ij}(D))$ to transform the k input signals $u_i(D)$ into the n output signals $x_i(D)$. Let t denote the time and let $g_{ij}(D) = \sum_{\ell \geq 0} g_{ij}(\ell) D^\ell$. Then $x_i(D) = \sum_{j=1}^{k} u_j(D) g_{ji}(D)$ and breaking this down to individual bits gives

$$x_i(t) = \sum_{j=1}^{k} \sum_{\ell=0}^{\deg g_{ji}(D)} u_j(t-\ell) g_{ji}(\ell). \tag{2.1}$$

In the diagrams that follow (differing slightly from the cover picture) the signals travel from left to right or from top to bottom. Figure 2.1 shows the basic elements appearing in the diagrams as parts of the circuitry. The signals on the wires are described either as individual bits, e.g. $a(i)$, at a given time i, or as the respective generating functions, e.g. $a(D) = \sum_i a(i) D^i$.

We shall call any circuitry consisting of k input lines, n output lines and a collection of the above elements *a realization* of the generator matrix $G(D)$, if the bits on the output wires are related to the bits on the input wires by the equations 2.1. Any realization of a generator matrix for a convolutional code C is called a *physical encoder* for C.

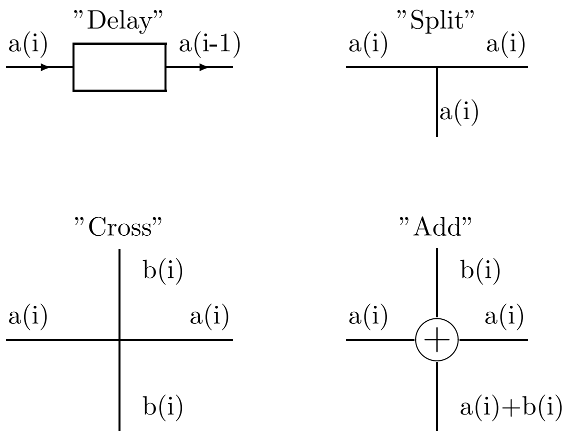


Figure 2.1: Parts of encoder circuits

To be able to compute the bit $x_i(t)$ from equation 2.1 we need to know all the bits $u_j(t-\ell)$, where ℓ ranges from 0 to e_j (= the maximum degree of the polynomials $g_{ji}(D)$). If we have a chain of e_j delayed copies of the signal $u_j(D)$ we can then compute $x_i(t)$ by including wires from suitable stages in the chains to the adders. This leads to the so called *direct-form realization* of $G(D)$.

We shall demonstrate the direct-form realization with all the generator matrices G_i of Example 1.5. Let us first take a look at the matrix G_1, see Figure 2.2. Reading these diagrams becomes easier after one makes the observation that on most diagrams all horizontal wires carry a signal of the form $D^i u_j(D)$ (this means that at time t the bit $u_j(t-i)$ is "on"). Thus in the left hand side of the first diagram we generate the signals $u_1(D), Du_1(D), D^2u_1(D)$ and $u_2(D), Du_2(D), D^2u_2(D)$ on the six horizontal wires. Then on the right hand side, the required sums are collected to the output wires.

From this construction it is obvious that $\operatorname{extdeg} G(D)$ delay elements suffice. Sometimes a clever observation will save a delay element. E.g. for the matrix G_1 we can make the observation that we never need the individual twice delayed bits $u_1(i-2)$ and $u_2(i-2)$. Only their sum is ever used in the computation of the output bits. So we save a single delay by first computing the sum $u_1(i-1) + u_2(i-1)$ and only after that introduce the second delay. This gives another realization of the matrix G_1, see Figure 2.3.

The generator matrices G_2 and G_3 only differ from G_1 in the second row. This also shows on the corresponding circuits

A similar saving as in the case of G_1 is possible for the matrix G_3 as here $u_2(i-1)$ and $u_1(i-2)$ only appear in the combination $u_1(i-2) + u_2(i-1)$. We leave it as an exercise to construct such a realization of the generator matrix G_3 that has only two delay elements. The canonical generator matrix G_4 (not surprisingly) yields the very simple circuit of Figure 2.6.

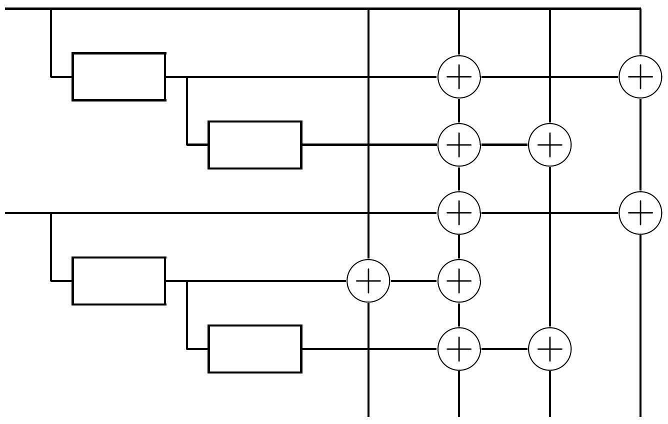


Figure 2.2: Direct-form realization of G_1

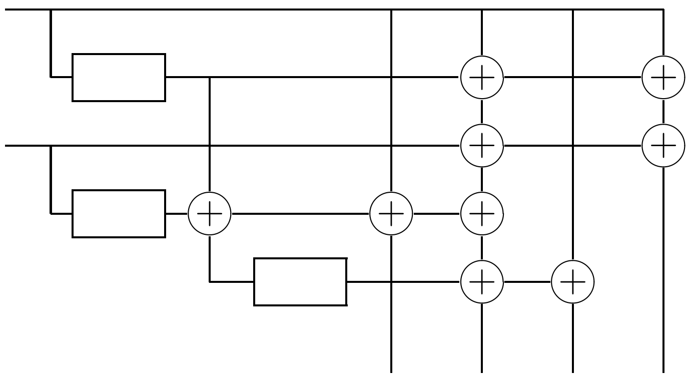


Figure 2.3: Another realization of G_1

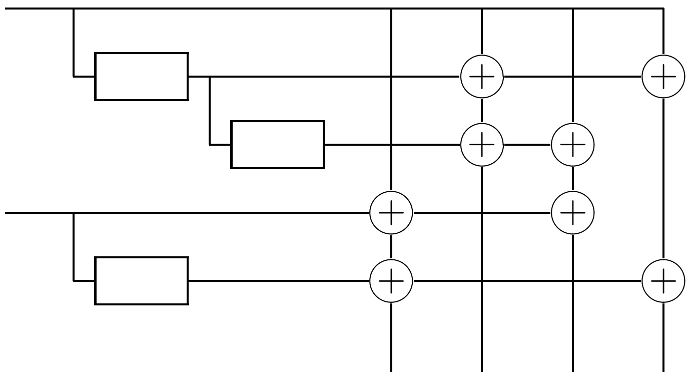


Figure 2.4: Direct-form realization of G_2

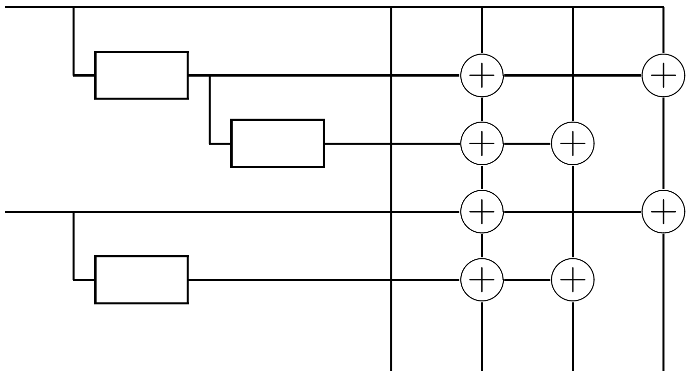


Figure 2.5: Direct-form realization of G_3

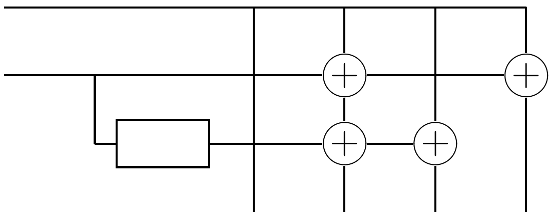


Figure 2.6: Direct-form realization of G_4

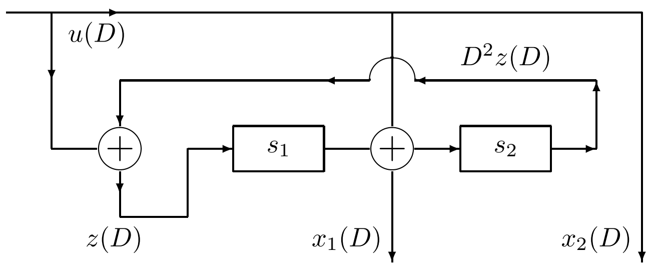


Figure 2.7: A realization of a systematic generator matrix

The minimum number of delay elements in a realization of a generator matrix $G(D)$ is called the *McMillan degree* $\operatorname{Mcdeg} G(D)$ of $G(D)$. The above examples suggest that for all polynomial generator matrices $G(D)$ the inequalities

$$\operatorname{intdeg} G(D) \leq \operatorname{Mcdeg} G(D) \leq \operatorname{extdeg} G(D)$$

might hold. The latter inequality is an immediate consequence of the direct-form construction. The former inequality turns out also to be true, but we shall skip the proof.

In addition to generating $\mathbf{F}[D]$-linear combinations of the input signals $u_i(D)$ one would like to be able to generate $\mathbf{F}(D)$-linear combinations. It is easy to see that this can be done, provided that the appearing rational functions are causal (i.e. in $\mathbf{F}(D) \cap \mathbf{F}[[D]]$). This requires the use of the so called *feedback*, i.e. feeding the output from a delay element to an earlier addition element.

For example, in Figure 2.7 we have a realization of a systematic generator matrix $G(D) = ((1 + D + D^2)/(1 + D^2), 1)$. Up to an exchange of the output wires this is the $(2, 1, 2)$ code from the introduction. To see this, let $z(D)$ be the signal leaving the first addition element. Studying the wires meeting at this addition element we may conclude that the equation

$$u(D) + D^2z(D) = z(D)$$

must hold. From this we easily solve that $z(D) = u(D)/(1 + D^2)$. It is then straightforward to see that the output signal $x_1(D)$ equals $u(D)(1 + D + D^2)/(1 + D^2)$ as claimed.

2.2 Abstract encoders

We refer to an encoder in the sense of definition 0.1 as an *abstract encoder* in contrast to the physical encoders that appeared in the previous section. In this section we show, how the geometry of a physical encoder describes an abstract encoder. Hence we have completed a proof of the equivalence of our two definitions of a convolutional code.

Given a realization of a generator matrix for an (n, k) code we let m be the number of delay elements in the physical encoder. We build the abstract encoder on the assumption that the state vector has, at any given time i, as components the contents of all the delay units in a certain fixed order.

We show this by studying (hopefully convincing) examples. Let us first study our first realization of the matrix G_1, see Figure 2.2. Let us number the delay elements 1,2,3 and

4 from top to bottom. If at a time i the delay elements contain the bits s_1, s_2, s_3 and s_4 respectively and bits u_1 and u_2 are fed to the two input lines, we see that the output bits x_1, x_2, x_3, x_4 are

$$\begin{aligned} x_1 &= u_1 + s_3, \\ x_2 &= u_1 + u_2 + s_1 + s_2 + s_3 + s_4, \\ x_3 &= u_1 + s_2 + s_4, \\ x_4 &= u_1 + u_2 + s_1. \end{aligned}$$

Thus we see that the matrices $\mathcal{C}$ and $\mathcal{D}$ used in computing the output vector $\mathbf{x}$ are

$$\mathcal{C} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \qquad \mathcal{D} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

Similarly we see that the contents of the delay elements are replaced by

$$\begin{aligned} s_1 &\to u_1, \\ s_2 &\to s_1, \\ s_3 &\to u_2, \\ s_4 &\to s_3. \end{aligned}$$

So the matrices $\mathcal{A}$ and $\mathcal{B}$ governing the time-evolution are

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \mathcal{B} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Similarly we see that the improved realization of the generator matrix G_1, see Figure 2.3, gives rise to the abstract $(4, 2, 3)$ encoder corresponding to the matrices

$$\begin{aligned} \mathcal{A} &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, & \mathcal{B} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \\ \mathcal{C} &= \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, & \mathcal{D} &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}. \end{aligned}$$

A realization of a causal rational generator matrix also leads easily to an abstract encoder. For example, let us consider the physical encoder of Figure 2.7. Let u be the input bit at a time, when the contents of the two memory chips are s_1 and s_2. We immediately see that the output bits are

$$\begin{aligned} x_1 &= u + s_1 \\ x_2 &= u \end{aligned}$$

and that the contents of the memory chips are replaced by

$$\begin{aligned} s_1 &\to u + s_2, \\ s_2 &\to s_1. \end{aligned}$$

These equations correspond to an abstract encoder given by the matrices

$$\begin{aligned} \mathcal{A} &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, & \mathcal{B} &= \begin{pmatrix} 1 & 0 \end{pmatrix}, \\ \mathcal{C} &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, & \mathcal{D} &= \begin{pmatrix} 1 & 1 \end{pmatrix}. \end{aligned}$$

It is an easy exercise to show that the direct form realization of a polynomial generator matrix leads to such an abstract encoder that the matrix $\mathcal{A}$ is nilpotent. The previous example shows that for a non-polynomial generator matrix this need not hold. In fact, it is not difficult to see that, starting with a realization of a non-polynomial generator matrix, the resulting matrix $\mathcal{A}$ cannot be nilpotent.

2.3 State space theorem

Our next goal is to show that the degree of a convolutional code is always a lower bound to the McMillan degree of any generator matrix. Our proof of this depends on certain ideas from linear system theory.

The basic idea comes from the observation that, for a physical encoder, the state, i.e. the contents of the delay elements, clearly determine the collection of possible futures of the encoder. Similarly, if we know the state vector of an abstract encoder at a particular time t, we can determine the set of possible output sequences $\mathbf{x}(i), i \geq t$. It may happen that for some encoders different values of the known state vector give rise to the same set of possible futures. However, one might hope that for a minimal encoder this wouldn't happen. This line of thought could lead one to define an *abstract state* to be a collection of possible futures of *the code* (rather than of an encoder). We shall not pursue this line of thought, but our proof of the state space theorem (due to Forney) does reflect this idea: namely it is natural to think that the zero vector in the space of abstract states should correspond to the set of possible futures, when the current state is equal to zero. For a convolutional code with a polynomial generator matrix (actually we may even allow rational entries as long as they are causal) this translates (at time $t = 0$) to the set of *causal codewords.*

Recall that we allow our codewords $\mathbf{x}(D) = \sum_{i \geq m} \mathbf{x}_i D^i \in \mathbf{F}^n((D))$ to begin at any time $m \in \mathbf{Z}$. We call such a word causal, if $\mathbf{x}_i = 0$ for any $i < 0$. For any power series $\mathbf{x}(D) = \sum_{i \geq m} \mathbf{x}_i D^i \in \mathbf{F}^n((D))$ we call

$$\mathbf{x}^+(D) = \sum_{i \geq \max\{0,m\}} \mathbf{x}_i D^i$$

the *causal part* of $\mathbf{x}(D)$. Naturally then $\mathbf{x}^-(D) = \mathbf{x}(D) - \mathbf{x}^+(D)$ is called the *anticausal part.*

Suppose that C is an (n, k) convolutional code with Forney indices $(e_1, e_2, \ldots, e_k)$ and degree $m = e_1 + e_2 + \cdots + e_k$. Let C^* be the subset of such codewords, whose causal part is also a codeword. Both C and C^* are vector spaces over the binary field $\mathbf{F}$ and clearly C^* is a subspace of C. We define the *abstract state space of* C, denoted Σ_C, to be the corresponding quotient space

$$\Sigma_C = C/C^*.$$

The main result of this chapter is then the following **State space theorem**.

Theorem 2.1 *If* Σ_C *is the abstract state space of a given* (n,k) *convolutional code of degree* m*, then*

$$\dim \Sigma_C = m.$$

Proof. Let $G(D)$ be a canonical generator matrix for C, with rows $\mathbf{g}_1(D), \mathbf{g}_2(D), \ldots, \mathbf{g}_k(D)$ in $F^n[D]$ and $\deg \mathbf{g}_i(D) = e_i$, for $i = 1, 2, \ldots k$. Now consider the following m codewords of C:

$$\mathbf{b}_{i,j}(D) = D^i \mathbf{g}_j(D), i = 1, \ldots, k, j = -1, -2, \ldots, -e_i.$$

We shall prove the theorem by showing that the elements $\mathbf{b}_{i,j}(D)$ (or rather their cosets modulo C^*) form a basis of Σ_C.

Let us first show that the cosets of $\mathbf{b}_{i,j}(D)$ span the entire state space. So let $\mathbf{x}(D) = \sum_{i=1}^{k} u_i(D)\mathbf{g}_i(D)$ be an arbitrary codeword in C, where $\mathbf{u}_i(D) = \sum_j u_{i,j}D^j, i = 1, 2, \ldots, k$ are then uniquely determined Laurent series. Let then

$$\begin{aligned} u_i'(D) &= \sum_{j=-e_i}^{-1} u_{i,j}D^j, \quad \text{for all } i = 1, 2, \ldots, k \text{ and} \\ \mathbf{x}'(D) &= \sum_{i=1}^{k} u_i'(D)\mathbf{g}_i(D). \end{aligned}$$

Then

$$\mathbf{x}'(D) = \sum_{i=1}^{k} \sum_{j=-e_i}^{-1} u_{i,j}\mathbf{b}_{i,j},$$

is in the span of the vectors $\mathbf{b}_{i,j}$, so it suffices to show that the codeword $\mathbf{x}(D) - \mathbf{x}'(D)$ is in the subspace C^*. But

$$\begin{aligned} \mathbf{x}(D) - \mathbf{x}'(D) &= \sum_{i=1}^{k}\sum_{j} u_{i,j}D^j\mathbf{g}_i(D) - \sum_{i=1}^{k}\sum_{j=-e_i}^{i} u_{i,j}D^j\mathbf{g}_i(D) \\ &= \sum_{i=1}^{k}\left(\sum_{j<-e_i} u_{i,j}D^j\right)\mathbf{g}_i(D) + \sum_{i=1}^{k}\left(\sum_{j\geq 0} u_{i,j}D^j\right)\mathbf{g}_i(D) \end{aligned}$$

is evidently a sum of an anticausal codeword and a causal codeword. This shows that $\mathbf{x}(D) - \mathbf{x}'(D) \in C^*$ and that $\dim C/C^* \leq \operatorname{extdeg} G(D)$.

We still need to show that the cosets of the vectors $\mathbf{b}_{i,j}$ are linearly independent in C/C^*. This is equivalent to proving that no non-zero $\mathbf{F}$-linear combination of the vectors $\mathbf{b}_{i,j}$ lies in the subspace C^*. Assume contrariwise that the causal part $\mathbf{x}^+(D)$ of

$$\mathbf{x}(D) = \sum_{i=1}^{k}\sum_{j=-e_i}^{-1} u_{i,j}\mathbf{b}_{i,j}(D), \quad u_{i,j} \in \mathbf{F},$$

is also a codeword. Then $\mathbf{x}^+(D)$ must be an $\mathbf{F}((D))$-linear combination

$$\mathbf{x}^+(D) = \sum_{i=1}^{k} v_i(D)\mathbf{g}_i(D), \quad v_i(D) \in \mathbf{F}((D)),$$

of the generators $\mathbf{g}_i(D)$. By the "polynomial output implies polynomial input" property of canonical matrices, the coefficients $v_i(D)$ must actually be polynomials.

Let us turn our attention to the anticausal part of $\mathbf{x}(D)$

$$\mathbf{x}^-(D) = \mathbf{x}(D) + \mathbf{x}^+(D) = \sum_{i=1}^{k} \left(v_i(D) + \sum_{j=-e_i}^{-1} u_{i,j} D^j \right) \mathbf{g}_i(D).$$

If any of the polynomials $v_i(D)$ is non-zero, then by the "predictable degree property" of canonical generator matrices, the degree of $\mathbf{x}^-(D)$ should be at least $e_i \geq 0$. As this is clearly a contradiction, we can conclude that $\mathbf{x}^+(D) = 0$, i.e. $\mathbf{x}(D)$ is anticausal. On the other hand, if any of the coordinates

$$u_i(D) = \sum_{j=-e_i}^{-1} u_{i,j} D^j$$

of the vector $\mathbf{x}(D)$ with respect to the basis $\{\mathbf{g}_i(D)\}$ is non-zero, hence of degree $\geq -e_i$, then again by the predictable degree property $\deg \mathbf{x}(D) \geq e_i - e_i = 0$. This contradicts the fact that $\mathbf{x}(D)$ was seen to be anticausal. ∎

Corollary 2.1 *In any physical encoder for an (n, k, m) convolutional code C the number of delay elements is at least m.*

Proof. Let us study an encoder containing r delay elements. We define a function $\pi : C \to \mathbf{F}^r$ that simply maps a codeword $\mathbf{x}(D)$ to the r-tuple consisting of the contents of the delay elements of the encoder at time $t = 0$ while generating $\mathbf{x}(D)$. This is obviously an $\mathbf{F}$-linear mapping.

Let us assume that $\mathbf{x}(D) \in \ker \pi$, i.e. when generating the output $\mathbf{x}(D)$, the encoder is at the zero state at time $t = 0$. But by definition, a codeword is the output of an encoder in response to some input, *assuming that initially the encoder is at state 0.* Hence the causal part $\mathbf{x}^+(D)$ of $\mathbf{x}(D)$ is a codeword — after all it is the response of the encoder to the inputs yielding $\mathbf{x}$ after time $t = 0$ and starting at the zero state. Thus $\mathbf{x}(D) \in C^*$ and $\ker \pi \subset C^*$.

Let $V \subset \mathbf{F}^r$ be the image of the mapping π. The result of the previous paragraph may be interpreted by stating that the function $f : V \to \Sigma_C$ that sends $\pi(\mathbf{x}(D))$ to the coset $\mathbf{x}(D) + C^*$ is a well-defined $\mathbf{F}$-linear map. It is obviously a surjection, so the dimension of V must be at least $\dim \Sigma_C = m$. On the other hand the dimension of V is at most r and the claim follows. ∎

Corollary 2.2 *If $G(D)$ is a canonical polynomial generator matrix for the code C, then*

$$\operatorname{Mcdeg} G(D) = \operatorname{extdeg} G(D).$$

Proof. From the previous Corollary we get that $\operatorname{Mcdeg} G(D) \geq \deg C = \operatorname{extdeg} G(D)$. The reverse inequality follows from the fact the direct-form realization of $G(D)$ has exactly $\operatorname{extdeg} G(D)$ delay elements. ∎

We have seen that the direct-form realization of a canonical generator matrix yields in a way the simplest possible physical encoder for the code. However, this is not the whole story — other polynomial generator matrices may lead to equally good physical encoders. Furthermore, an immediate corollary to the theorem below is that a systematic generator matrix always has a minimal McMillan degree.

Example 2.1 Show that the matrix

$$G_6 = \begin{pmatrix} 1 & D & 1+D & 0 \\ 0 & 1+D & D & 1 \end{pmatrix}$$

is basic but not reduced. Furthermore, show that it generates the code of Example 1.5 and that $\operatorname{Mcdeg} G_6 = 1$.

We state without proof a result due to Forney that settles the question, when a generator matrix has minimal possible McMillan degree.

Theorem 2.2 *Let $G(D)$ be a polynomial generator matrix for an (n, k) convolutional code C. Then* $\operatorname{Mcdeg} G(D) = \deg C$, *iff $G(D)$ has both a polynomial right inverse and an antipolynomial (i.e., polynomial in D^{-1}) right inverse.*

In other words, a necessary and sufficient condition for $G(D)$ to have minimal McMillan degree is the existence of matrices $H_1(D) \in \mathcal{M}_{n\times k}(\mathbf{F}[D])$ and $H_2(D) \in \mathcal{M}_{n\times k}(\mathbf{F}[D^{-1}])$ such that

$$G(D)H_1(D) = G(D)H_2(D) = I_k.$$

For example the matrix G_6 of the previous example has a polynomial right polynomial inverse, because it is basic. It is also easy to verify that

$$H_2(D) = \begin{pmatrix} 0 & 0 \\ D^{-1} & 0 \\ 0 & 0 \\ 1+D^{-1} & 1 \end{pmatrix}$$

is an antipolynomial right inverse of G_6.

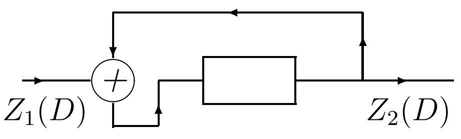
Exercises

2.1 *Draw the direct-form realization of the matrix G_2 from exercise 1.7. Find the corresponding abstract encoder.*

2.2 *Find a realization of the generator matrix G_3 from example 1.5 that needs only two delay elements.*

2.3 *It is possible to give realizations for some non-polynomial generator matrices as well, but this requires the use of feedback.*

A) *Show that the signals $Z_1(D)$ and $Z_2(D)$ in the following circuit are related to each other by the equation $Z_2(D) = DZ_1(D)/(1+D)$.*



B) *The matrix*

$$G_5 = \begin{pmatrix} 1 & 0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0 & 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix}$$

generates the code of example 1.5. Design such a circuit realizing the matrix G_5 that uses only a single delay element (and feedback).

2.4 *Let $G(D)$ be a polynomial generator matrix for an (n,k) code C. Let the row degrees of $G(D)$ be $e_1, e_2, \ldots, e_k$. Let us construct an abstract encoder from the direct-form realization of $G(D)$ as in section 2.2. Convince yourself that the components of the resulting state vector at time t are $u_i(t-j)$, where j ranges from 1 to e_i and i ranges from 1 to k. Show that the matrix $\mathcal{A}$ of this abstract encoder is nilpotent. Hint: The power $\mathcal{A}^r$ governs the time evolution of the state vector over a time interval of length r* in response to a sequence of zero inputs.

2.5 *Let C be an (n,k) convolutional code and $m \in \mathbf{Z}$. Let us define the following subspaces of C*

$$\begin{aligned} C_{<m} &= \{\mathbf{x}(D) = \textstyle\sum_i \mathbf{x}_i(D) \in C \mid \mathbf{x}_i = 0, \text{for all } i \geq m\}, \\ C_{\geq m} &= \{\mathbf{x}(D) = \textstyle\sum_i \mathbf{x}_i(D) \in C \mid \mathbf{x}_i = 0, \text{for all } i < m\}. \end{aligned}$$

Show that the subspace C^* described in section 2.3 is actually the direct sum

$$C^* = C_{<0} \oplus C_{\geq 0}.$$

2.6 *Use the notation of the previous problem. Show that for all i the following vector spaces $\Sigma_{C,i}$ (abstract state space at time i) all have the same dimension (= the degree of the code)*

$$\Sigma_{C,i} = C/\left(C_{<i} \oplus C_{\geq i}\right).$$

2.7 *Theorem 2.2 guarantees that a systematic polynomial generator matrix always has a minimal McMillan degree (why?). In exercise 1.5 we saw that the code generated by*

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

has degree 2. Find a realization of $G(D)$ which has only two delay elements.

2.8 *Let C be an (n,k) convolutional code with Forney indices $e_1 \leq e_2 \cdots \leq e_k$. Let m be any integer with the property that $m \geq e_k$. With the notation of exercise 2.6 show that*

$$C = C_{<m} + C_{\geq 0}.$$

Is this a direct sum? (here '+' is a sum of vector spaces over $\mathbf{F}$).

2.9 *Let $G(D)$ be a canonical generator matrix for an (n,k) convolutional code C and let the abstract encoder corresponding to the direct-form realization of $G(D)$ be given by the matrices $\mathcal{A},\mathcal{B},\mathcal{C}$ and $\mathcal{D}$. Show that the matrix $\mathcal{D}$ is gotten from $G(D)$ by replacing D with zero, i.e. $\mathcal{D}=G(0)$. Conclude that $\mathcal{D}$ has full rank.*

In the following problems we will prove a partial result related to Theorem 2.2 and study the effect of a change in the direction of time. Throughout $G(D)$ is a given canonical generator matrix for an (n,k) code and $\mathbf{g}_1(D),\ldots,\mathbf{g}_k(D)$ are its rows. The respective degrees of the row vectors are assumed to be $e_1,e_2,\ldots,e_k$. We will study the matrix $\tilde{G}(D)$ that has rows $\tilde{\mathbf{g}}_i(D)=D^{e_i}\mathbf{g}_i(D^{-1})$. It is obviously a polynomial generator matrix for another (n,k) convolutional code $\tilde{C}$. We denote by $\bar{G}$ and $\bar{\tilde{G}}$ the indicator matrices for the highest degree terms of $G(D)$ and $\tilde{G}(D)$ (cf. Theorem 1.3).

2.10 *Show that $\bar{\tilde{G}}=G(0),\bar{G}=\tilde{G}(0)$ and conclude that $\tilde{G}(D)$ is reduced.*

2.11 *Show that $\tilde{G}(D)$ is basic and hence also canonical. Conclude that the Forney indices of C and $\tilde{C}$ are equal. Hint: Theorem 1.2 suggests at least two different ways of showing this.*

2.12 *Show that $G(D)$ has an antipolynomial right inverse.*

2.13 *Show that a* finite weight *codeword $\mathbf{x}(D)\in\mathbf{F}^n((D))$ is in $\tilde{C}$, iff $\mathbf{x}(D^{-1})$ is in C. Conclude that the free distances of C and $\tilde{C}$ are equal. Is the finiteness assumption of the weight of $\mathbf{x}(D)\in\tilde{C}$ necessary for $\mathbf{x}(D^{-1})$ to be in C?*

Chapter 3

Graphical presentations of a convolutional code

Some aspects of the theory of convolutional codes can be best illustrated with the aid of suitable graphs. We shall give two different graphical presentations, namely *state diagrams* and *trellis diagrams*. We will use the former to introduce an algorithm for computing certain generating functions. As a consequence we also get an algorithm for computing the free distance of a convolutional code. The trellis diagrams will give an easy visualization of the so called *Viterbi decoding algorithm*.

Both the state and trellis diagrams are presentations of a convolutional encoder rather than of a convolutional code. The size of the diagrams (as well as the complexity of the above mentioned algorithms) depends exponentially on the dimension of the state space of the encoder. This again underlines the importance of canonical generator matrices.

3.1 State diagrams of encoders

Let us fix an abstract encoder $(\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D})$ of an (n,k,m) convolutional code. The state diagram of such an encoder is the following labelled directed graph $\mathcal{G}$: There are 2^m vectors in the state space, each uniquely identified by an m-tuple of bits. We let the state vectors be the vertices of $\mathcal{G}$. There are 2^k edges of $\mathcal{G}$ leaving any given vertex. They are in a one-to-one correspondence with the input vectors and they represent the state transitions: given an input vector $\mathbf{u} \in \mathbf{F}^k$ and a state $\mathbf{s} \in \mathbf{F}^m$, in response to the input $\mathbf{u}$ the encoder will move from state $\mathbf{s}$ to the state

$$\mathbf{s}' = \mathbf{s}\mathcal{A} + \mathbf{u}\mathcal{B}.$$

We express this in the state diagram by connecting the vertex $\mathbf{s}$ to the vertex $\mathbf{s}'$ with an arrow labelled with two vectors $\mathbf{u}/\mathbf{x}$: the input $\mathbf{u}$ and the corresponding output

$$\mathbf{x} = \mathbf{s}\mathcal{C} + \mathbf{u}\mathcal{D}.$$

The state diagram of the $(2,1,2)$ encoder in the introduction is shown in Figure 3.1.

As the state diagram faithfully captures the state transitions and outputs that result from a given input, the encoder (and hence also the code) is completely determined from the state diagram. For example, we see that codewords can be reconstructed as sequences of output labels along such continuous paths through the state diagram that begin and end at the

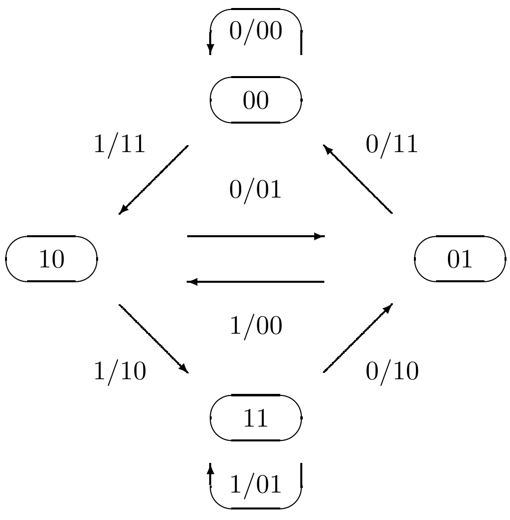


Figure 3.1: State diagram of a $(2,1,2)$ encoder of Example 1.3

node corresponding to the zero state (= zero vertex). If we agree that travelling along a single edge of such a path takes one unit of time, then only the starting point on the time axis (which can be chosen at will anyway) is lost.

We will call (for lack of a better term) a codeword *simple*, if the corresponding path through the state diagram passes by the zero vertex only in the beginning and in the end. Obviously any codeword can be expressed as such a sum of simple codewords that are being output at disjoint intervals of time — simply cut the corresponding closed path in the state diagram to pieces at such moments, when the path passes by the zero vertex.

Our goal will be to compute a generating function that tells us the number of simple codewords of a given output weight i, input weight j and time span (= path length in the state diagram) k. If the number of such codewords is $A(i,j,k)$ the following generating function, called the *transfer function of the encoder* captures all this information

$$T(X,Y,Z) = \sum_{i,j,k} A(i,j,k) X^i Y^j Z^k \in \mathbf{Z}[[X,Y,Z]].$$

In order to count simple codewords we modify the state diagram somewhat. We split the zero state into an initial zero state $(\bar{0},i)$ and a final zero state $(\bar{0},f)$. The edges are split such that the initial zero state has no incoming edges and the final state has no outgoing edges. We omit the the zero loop going from the zero vertex to itself altogether. Also the labels are changed: we simply label an edge with a monomial $X^i Y^j Z^k$, where the exponents i,j,k are the output weight, the input weight and the path length increment (=1) respectively. For the encoder of Example 1.3 this resulting *modified state diagram* is shown in Figure 3.2.

We use this modified diagram to compute the transfer function. First we need the corresponding generating functions counting such incomplete paths that end in a given vertex. These are the following intermediate transfer functions $\psi_{\mathbf{s}}$ defined for all vertices $\mathbf{s}$ in the modified state diagram

$$\psi_{\mathbf{s}}(X,Y,Z) = \sum_{i,j,k} A_{\mathbf{s}}(i,j,k) X^i Y^j Z^k \in \mathbf{Z}[[X,Y,Z]],$$

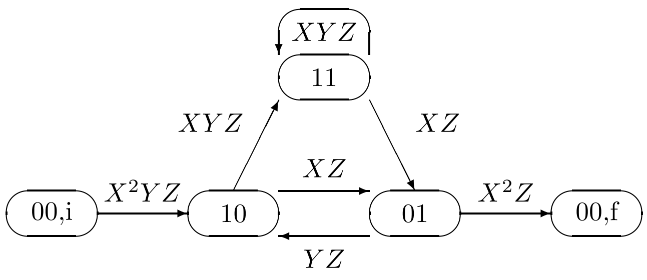


Figure 3.2: Modified state diagram of a $(2, 1, 2)$ encoder

where $A_{\mathbf{s}}(i, j, k)$ is the number of such paths from the initial zero to $\mathbf{s}$ in the modified state diagram that have output weight i, input weight j and length k. Thus $\psi_{\bar{0},i}(X, Y, Z) = 1$ and $\psi_{\bar{0},f}(X, Y, Z) = T(X, Y, Z)$.

We see that recursive formulas for the quantities $A_{\mathbf{s}}(i, j, k)$ can be derived from a study of the modified state diagram. Obviously $A_{\bar{0},i}(0, 0, 0) = 1$ and $A_{\mathbf{s}}(i, j, k) = 0$, if any of the exponents i, j, k is negative and $A_{\bar{0},i}(i, j, k) = 0$, if $(i, j, k) \neq (0, 0, 0)$. We get recurrence relations for the remaining numbers $A_{\mathbf{s}}(i, j, k)$ as follows. A path contributing to the number $A_{\mathbf{s}}(i, j, k)$ must come to the vertex $\mathbf{s}$ from a neighboring vertex. We can divide such paths into groups based on what is the last edge in the path. Let the label of an edge e be $X^{i(e)}Y^{j(e)}Z$ and the starting vertex (resp. final vertex) of e be $s(e)$ (resp. $f(e)$). As all the paths belong to one and only one group, for all $\mathbf{s}, i, j, k$ we have

$$A_{\mathbf{s}}(i, j, k) = \sum_{e, f(e)=\mathbf{s}} A_{s(e)}(i - i(e), j - j(e), k - 1). \tag{3.1}$$

Clearly the equations 3.1 always allow us to recursively compute all the numbers $A_{\mathbf{s}}(i, j, k)$. The reason for this is that in the above equation, the terms in the right hand side involve only such quantities $A_{\mathbf{s}'}(i', j', k')$, where $i + j + k > i' + j' + k'$.

In order to find the intermediate transfer functions we multiply the equation 3.1 by $X^iY^jZ^k$ and sum over all i, j, k and arrive at the equations

$$\psi_{\mathbf{s}}(X, Y, Z) = \sum_{e, f(e)=\mathbf{s}} X^{i(e)}Y^{j(e)}Z\psi_{s(e)}(X, Y, Z), \tag{3.2}$$

for all the vertices $\mathbf{s}$. From this we can solve the functions $\psi_{\mathbf{s}}$ and hence also the transfer function $T(X, Y, Z)$.

For example from Figure 3.2 we get the following system of equations

$$\begin{cases} \psi_{10} &= X^2YZ + YZ\psi_{01} \\ \psi_{01} &= XZ\psi_{10} + XZ\psi_{11} \\ \psi_{11} &= XYZ\psi_{11} + XYZ\psi_{10} \\ \psi_{00,f} &= X^2Z\psi_{01}. \end{cases}$$

From this we can solve that

$$T(X, Y, Z) = \psi_{00,f}(X, Y, Z) = \frac{X^5YZ^3}{1 - XYZ(1 + Z)}.$$

The transfer function contains a little bit more information than what is required to compute the free distance. If we want a generating function that simply enumerates the simple codewords of a given weight, we can do this by simply substituting $Y = 1$ (don't care about the input weight) and $Z = 1$ (don't care about the path length). In the above example we get

$$T(X,1,1) = \frac{X^5}{1-2X} = X^5 + 2X^6 + 4X^7 + \cdots$$

that confirms our earlier computation showing that the free distance of this code is equal to 5. We also see that there are two different simple codewords of weight 6, four of weight 7 etc.

A word of warning is in order here: It is by no means clear that we can always find $T(X,1,1)$ and the free distance from the above computations, if we use a poorly chosen encoder. Our argument showing that the recurrence relations 3.1 always allows to compute the numbers $A_{\mathbf{s}}(i,j,k)$ (and hence the function $\psi_{\mathbf{s}}(X,Y,Z)$) depended on the fact that the exponent of Z always increases, as we travel along an edge. If we set $Y = Z = 1$, then the previous argument is no longer valid. Indeed, for a bad encoder the recursion may not work and infinities may arise.

For example, if the modified state diagram contains a cycle consisting of edges of zero output weight, then obviously for some values of i the number of simple codewords of weight i is infinite. It is not too difficult to see that the presence of cycles of zero output weight is actually the only thing that may prevent recursive application of the equations 3.1 from yielding finite values for all the coefficients of the series $T(X,1,1)$.

Proposition 3.1 *Let $\mathcal{G}$ be the modified state diagram of a convolutional encoder. Let $\mathbf{s}$ be a vertex of $\mathcal{G}$ and let $A_{\mathbf{s}}(i) = \sum_{j,k\ge 0} A_{\mathbf{s}}(i,j,k)$ be the number of such paths in $\mathcal{G}$ that begin from $\bar{0}_i$ end at $\mathbf{s}$ and have total output weight i. Assume that $\mathcal{G}$ has no cycles consisting of edges with zero output labels, i.e. such labels, where the exponent of X is zero. Then the numbers $A_{\mathbf{s}}(i)$ are finite for all $i \ge 0$ and all vertices $\mathbf{s}$. Furthermore, these quantities can be determined from the equations*

$$A_{\mathbf{s}}(i) = \sum_{e, f(e)=\mathbf{s}} A_{s(e)}(i - i(e)). \tag{3.3}$$

In particular the series $T(X,1,1) \in \mathbf{Z}[[X]]$.

Proof. We shall sketch an argument. The equations 3.3 can be obtained by summing the equations 3.1 over the variables j and k. We say that the above equation is *centered* at the vertex $\mathbf{s}$. Let us study these equations thinking of the quantities $A_{\mathbf{s}}(i)$ as unknowns. We want to rewrite the equation 3.3 in such a form that on the right hand side only such quantities $A_{\mathbf{s}'}(i')$ appear that $i > i'$. If we can do this, then obviously our result follows by induction on i. However, it may happen that $i(e) = 0$ for some edge e that ends at $\mathbf{s}$. So we replace such terms $A_{s(e)}(i)$ by the right hand sides of the corresponding equations centered at $s(e)$. It may happen that other quantities $A_{\mathbf{s}'}(i)$ then appear, but we then repeat the procedure of replacing such terms with the right hand sides of the equation centered at $\mathbf{s}'$. As $\mathcal{G}$ has only a finite number of vertices and no cycles consisting of zero output edges, the process must terminate at some point. ∎

We close our study of this problem by proving the following result stating that for the direct-form realization (or for the corresponding abstract encoder) of a canonical generator matrix nothing bad will happen.

Theorem 3.1 *Let $G(D)$ be a canonical generator matrix for an (n, k, m) convolutional code. Let the associated abstract encoder of the direct-form realization of $G(D)$ be determined by the matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ and let $\mathcal{G}$ be the modified state diagram of this encoder. Then*

1) The matrix $\mathcal{A}$ is nilpotent.

2) The graph $\mathcal{G}$ is connected.

3) The graph $\mathcal{G}$ contains no cycles consisting entirely of edges with zero weight output labels.

4) The number of simple codewords of a given finite weight is finite.

Proof. • 1) This is left as an exercise.
• 2) The state vector of the direct-form realization at time t contains the bits $u_i(t-j)$ where j ranges from 1 to e_i and i ranges from 1 to k. Obviously then a suitable sequence of input vectors fills the components of the state vector with any chosen bits. Thus every vertex of $\mathcal{G}$ can be reached from the initial zero vertex (or from any other vertex except the final zero).
• 3) Let us assume that $\mathcal{G}$ contains a zero output cycle. Let $\mathbf{s}$ be one of the vertices belonging to the cycle. Obviously the final zero vertex is not included in the cycle. By part 2 the encoder will reach state $\mathbf{s}$ as a response to some finite sequence of inputs. Continuing a path starting at time $i = 0$ from the initial zero vertex to $\mathbf{s}$ by repeating the cycle ever after, we can then find a finite weight codeword, whose path never enters the final zero vertex. By the "polynomial output implies polynomial input" property of canonical matrices, the inputs leading the encoder along this path must eventually be all zero vectors. In particular, the input vectors in the cycle must all be zero vectors. However, by part 1, the encoder will in response to a sufficient number of zero inputs eventually move to the zero state. This contradicts our construction of the path and thus proves part 3.
• 4) This now follows from part 3 and the previous proposition. ∎

3.2 Trellis diagrams of encoders

Another graphical presentation of an (n, k, m) encoder $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$ is its *trellis diagram*. It is also a directed graph and contains the same information in a slightly different lay-out: the vertices of the graph are pairs $(\mathbf{s}, i)$ consisting of a state $\mathbf{s}$ and a moment of time i. Thus there are (in theory) infinitely many vertices corresponding to the state $\mathbf{s}$. The edges correspond again to the state transitions of the encoder and their labels are as in the state diagram. However, in a trellis diagram the edges also keep track of time and thus for all i we have an edge labelled with the input $\mathbf{u}$ and the output $\mathbf{x} = \mathbf{s}\mathcal{C} + \mathbf{u}\mathcal{D}$ going from the vertex $(\mathbf{s}, i)$ to the vertex $(\mathbf{s}\mathcal{A} + \mathbf{u}\mathcal{B}, i + 1)$. In a trellis diagram the horizontal direction is thought of as the time axis and all vertices $\{(\mathbf{s}, i) \mid \mathbf{s} \in \mathbf{F}^m\}$ (constant i) are stacked on top of each other at a location corresponding to time i. The trellis diagram of our favorite $(2, 1, 2)$ encoder is shown in Figure 3.3.

We see that the trellis consists of several repeating blocks called *trellis sections* (the vertices at two consecutive moments of time, i and $i + 1$ and the edges between those vertices). A single trellis section obviously completely determines the encoder. Often a

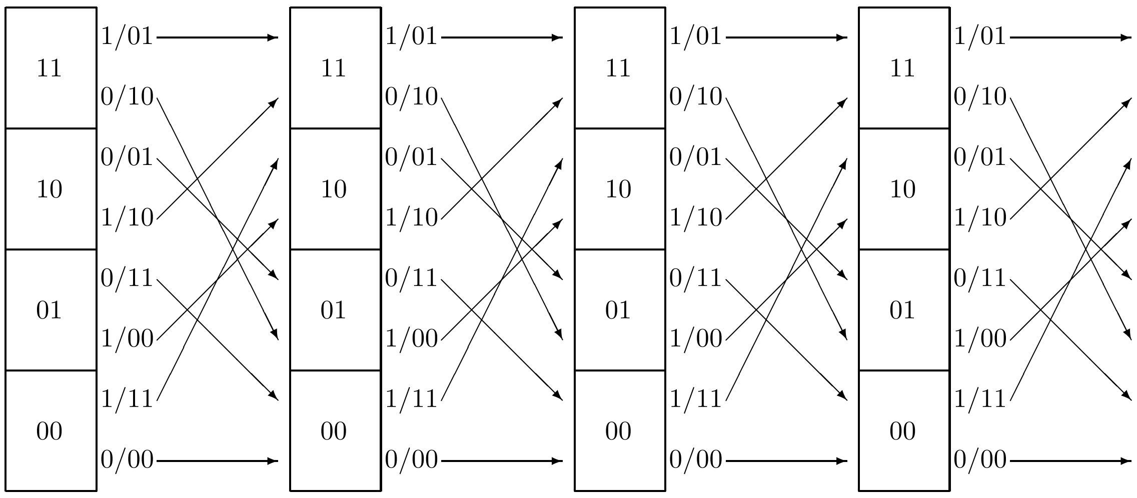


Figure 3.3: Trellis diagram of a $(2, 1, 2)$ encoder

trellis diagram is drawn in such a way that at the beginning of the transmission, say $i = 0$, only the zero vertex is shown. Then at later moments only those states that the encoder may have reached are shown. Thus for example in Figure 3.3 at time $i = 0$ only the 00-vertex would be there, at time $i = 1$ also the 10-vertex would be included and only at later times $i \geq 2$ would we have a complete trellis section. Similar reductions in the trellis occur also at the end of the transmission, where only the edges corresponding to the terminating zeros and the states such edges pass through are included in the diagram.

3.3 Viterbi decoding

As the trellis diagram of a convolutional encoder faithfully presents all the state transitions, the codewords are again such continuous paths through the trellis diagram that begin and end at the zero state. The trellis (and state diagram) presentation allows us to construct codewords small sections at a time. The Viterbi algorithm allows us to also *decode* a received word small sections at a time. This is quite different from usual decoding algorithms of block codes, like the Berlekamp–Massey algorithm for decoding the BCH codes, where it is necessary to wait for the entire codeword before decoding can begin.

We say that the Hamming distance $d(\mathbf{x}, \mathbf{y})$ of two vectors $\mathbf{x}(D), \mathbf{y}(D) \in \mathbf{F}((D))$ is the weight of their difference (if finite). When a convolutional code with a generator matrix $G(D)$ is used, the information $\mathbf{u}(D)$ is first encoded to $\mathbf{x}(D) = \mathbf{u}(D)G(D)$, which is then transmitted. Let us assume that at the receiving end the vector $\mathbf{y}(D)$ is observed. Due to errors caused by the channel it may be that $\mathbf{x}(D) \neq \mathbf{y}(D)$. The decoding problem is to try to find $\mathbf{u}(D)$ given that $\mathbf{y}(D)$ was received. If we assume that $G(D)$ is a canonical (hence basic) polynomial generator matrix, then it suffices to find $\mathbf{x}(D)$ as in this case Theorem 1.2 states that $G(D)$ has a right inverse $H(D)$ and $\mathbf{u}(D)$ can be recovered as $\mathbf{u}(D) = \mathbf{x}(D)H(D)$.

Under certain reasonable assumptions this amounts to finding the codeword $\mathbf{x}(D)$ that minimizes the Hamming distance $d(\mathbf{x}(D), \mathbf{y}(D))$ from the received word $\mathbf{y}(D)$. In a practical application we can assume that the times, when the transmission begins and ends, are known. This suggests that one should use a polynomial generator matrix (as we always do) and requires that so called *terminating zeros*, dummy zero input vectors, are being input to the encoder after the actual message $\mathbf{u}(D)$ has been completed. Recall that $\mathbf{x}(D)$ nearly always has higher degree than $\mathbf{u}(D)$ and thus more than $\deg \mathbf{u}(D)$ time units are required to transmit all of $\mathbf{x}(D)$. Thus we can assume that the encoder reaches the zero state at the end of transmission, i.e. that $\mathbf{x}(D)$ corresponds to a continuous path from $(\bar{0}, 0)$ to $(\bar{0}, N)$ in the trellis diagram, where N is the largest possible degree of $\mathbf{x}(D)$.

Let us assume that we have at hand a trellis diagram representing the code between times $i = 0$ and $i = N$, i.e. N sections of the trellis. We can then give each edge in the trellis graph a *cost* (in a graph theory course this was called *weight*, however, we have reserved that word for other use) as follows. Let us assume that $\mathbf{y}(D) = \sum_i \mathbf{y}_i D^i$ was received. The cost of an edge with output label $\mathbf{x}_i$ going from a vertex $(\mathbf{s}, i)$ to another vertex $(\mathbf{s}', i+1)$ is the Hamming distance $d(\mathbf{x}_i, \mathbf{y}_i)$ (this is the usual Hamming distance of binary vectors). As

$$d(\mathbf{x}(D), \mathbf{y}(D)) = \sum_{i=0}^{N} d(\mathbf{x}_i, \mathbf{y}_i),$$

we see that the Hamming distance between a codeword $\mathbf{x}(D) = \sum_i \mathbf{x}_i D^i$ and the received word $\mathbf{y}(D)$ is the total cost of the path through the trellis corresponding to the word $\mathbf{x}(D)$. Our decoding problem is thus reduced to the graph theoretical problem of finding a minimal cost path through a connected directed graph. In a graph theory course this was achieved by the so called Dijkstra's algorithm. When the graph is actually a trellis (not just any graph), Dijkstra's algorithm can be simplified drastically by taking advantage of the regularity of the trellis as a graph. Anyway, we will borrow from Dijkstra's algorithm the idea that one can find the minimal path connecting two given vertices A and B by recursively finding minimal paths from A to all the intermediate vertices.

The following lemma justifies the recursive step in the Viterbi decoding algorithm. We let $w(\mathcal{P})$ denote the total cost of a path $\mathcal{P}$ through the trellis diagram. We will denote by $\mathcal{P}e$ the path obtained by continuing the path $\mathcal{P}$ by the edge e. Whenever we use such a notation, it is implied that the end vertex of path $\mathcal{P}$ is also the starting vertex of e. The individual edges appearing in a path $\mathcal{P}$ are referred to as the *legs* of $\mathcal{P}$.

Lemma 3.1 *Assume that the minimal cost paths $\mathcal{P}_\mathbf{s}$ from $(\bar{0}, 0)$ to all the vertices $(\mathbf{s}, i), \mathbf{s} \in \mathbf{F}^m$ have been found for some $i \geq 0$ and that the respective costs of these paths are $w(\mathbf{s})$. Let $e_1, e_2, \ldots, e_r$ be all the edges that end at a given vertex $(\mathbf{s}, i+1)$. Let $(\mathbf{s}_j, i)$ be the starting vertex of the edge e_j and let w_j be the cost of the edge e_j. Then the minimal cost of a path from $(\bar{0}, 0)$ to $(\mathbf{s}, i+1)$ is the minimum w of the sums $w(\mathbf{s}_j) + w_j$ and $\mathcal{P}_{\mathbf{s}_j} e_j$ is a minimal cost path to, when j is chosen such that $w = w_j + w(\mathbf{s}_j)$.*

Proof. The construction obviously gives a path of the prescribed cost, so we only need to show that no path from $(\bar{0}, 0)$ to $(\mathbf{s}, i+1)$ can have a lower cost. Assume that $\mathcal{P}$ is such a path and that its cost is $w(\mathcal{P}) < w$. All the paths through trellis that end at $(\mathbf{s}, i+1)$ must have one of the edges e_j as the last leg in the path — no other incoming edges exist. So let us choose j_0 such that e_{j_0} is the last leg in $\mathcal{P}$, i.e. $\mathcal{P}$ can be written in the form $\mathcal{P} = \mathcal{P}' e_{j_0}$, where the path $\mathcal{P}'$ consists of all the earlier legs of $\mathcal{P}$. The path $\mathcal{P}'$ ends at the vertex $(\mathbf{s}_{j_0}, i)$

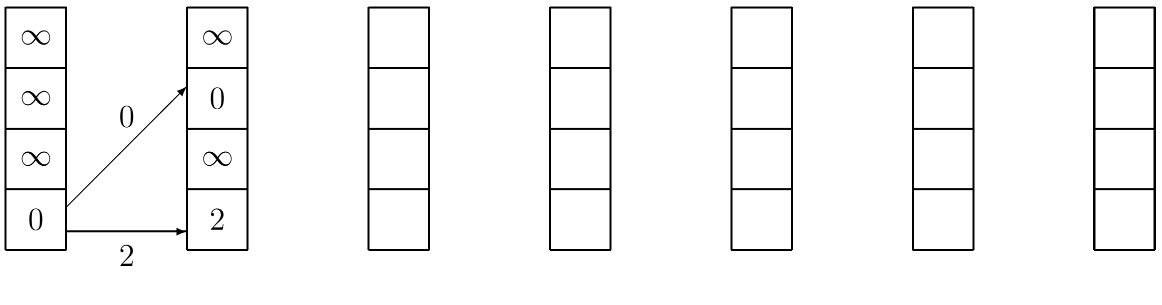
and thus by our assumption $\mathcal{P}'$ has cost $w(\mathcal{P}') \geq w(\mathbf{s}_{j_0})$. However, then the cost of $\mathcal{P}$ would be $w(\mathcal{P}) = w(\mathcal{P}') + w_{j_0} \geq w(\mathbf{s}_{j_0}) + w_{j_0} \geq w$, which is a contradiction. ∎

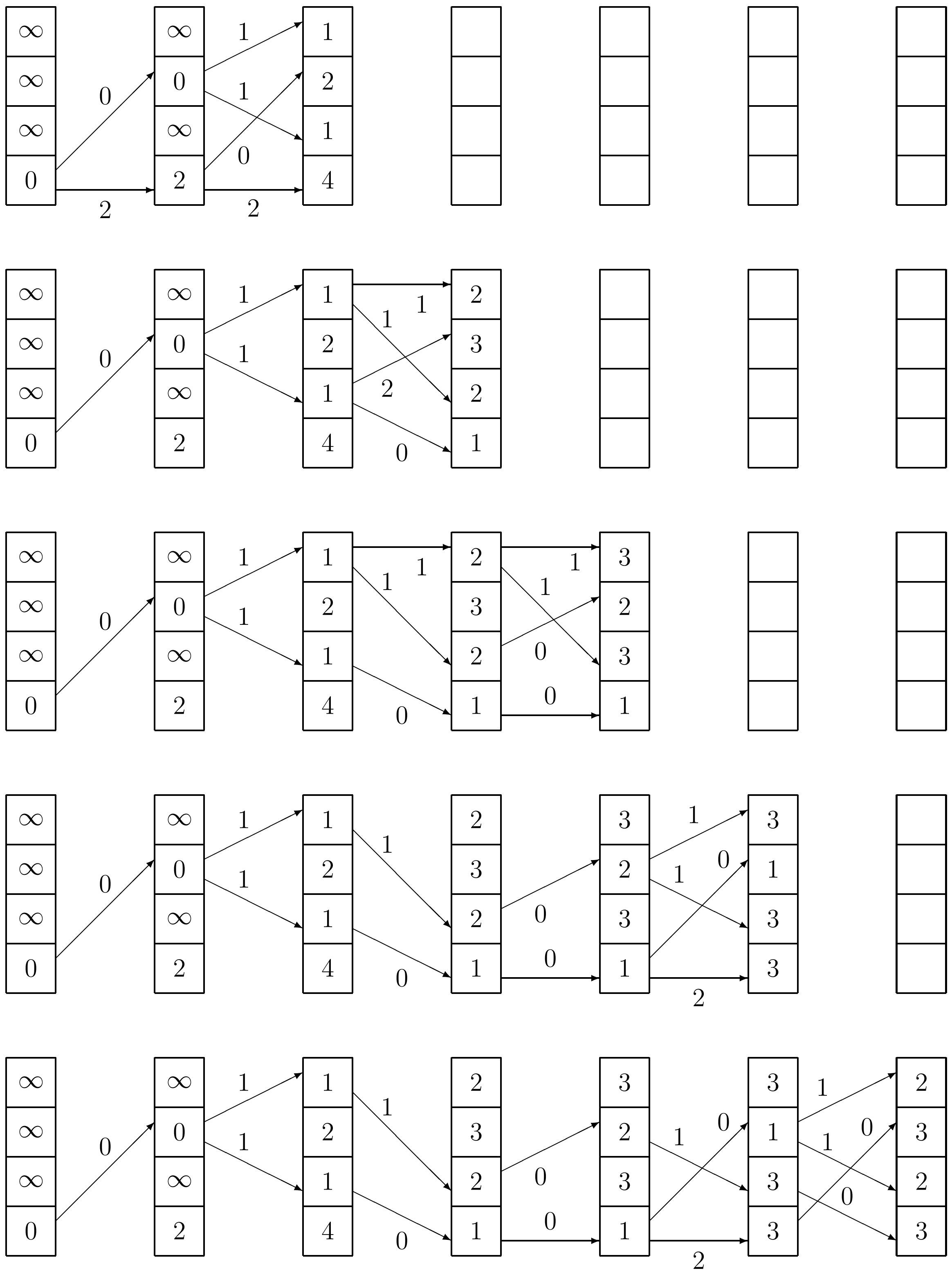
The Lemma 3.1 immediately leads to the following recursive algorithm for finding a minimal cost path from the starting vertex $(\bar{0}, 0)$ to the final vertex $(\bar{0}, N)$. At the starting point $i = 0$ we obviously have $w(\bar{0}, 0) = 0$ obtained by the empty path and $w(\mathbf{s}, 0) = \infty$ for all other vertices $(\mathbf{s}, 0), \mathbf{s} \neq \bar{0}$.

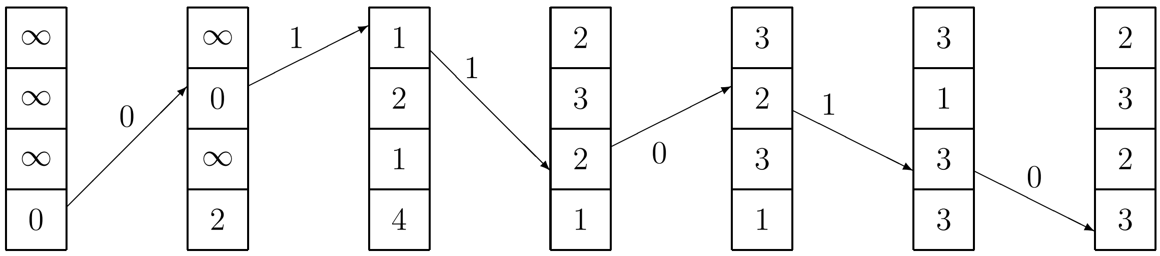
Viterbi decoding algorithm

```
Step 1. Initialize
    • $i := 0$
    • $w(\bar{0}, 0) := 0$
    • $w(\mathbf{s}, 0) := \infty$ for all $\mathbf{s} \in \mathbf{F}^m, \mathbf{s} \neq \bar{0}$
    • $\mathcal{P}(\bar{0}, 0) :=$ empty path
Step 2. For all $\mathbf{s} \in \mathbf{F}^m$ initialize $w(\mathbf{s}, i+1) := \infty$.
Step 3. For all $\mathbf{s} \in \mathbf{F}^m$ such that $w(\mathbf{s}, i) < \infty$ and all the edges $e$ leaving the vertex $(\mathbf{s}, i)$ do
    • Let $\mathbf{s}'$ be the end state of $e$.
    • If $w(e) + w(\mathbf{s}, i) < w(\mathbf{s}', i+1)$ then
        – $\mathcal{P}(\mathbf{s}', i+1) := \mathcal{P}(\mathbf{s}, i)e$
        – $w(\mathbf{s}', i+1) := w(e) + w(\mathbf{s}, i)$
Step 4. $i := i + 1$
Step 5. If $i < N$, then go to Step 2.
Step 6. Output $\mathcal{P}(\bar{0}, N)$.
```

We illustrate the Viterbi decoding algorithm by the following example. We use the $(2, 1, 2)$ code of the introduction, so the trellis is shown in Figure 3.3. The encoder giving this trellis corresponds to the generator matrix $G = (1 + D^2, 1 + D + D^2)$. This matrix has a right inverse $H = \binom{1+D}{D}$. Assume that $N = 6$ and that we receive the following sequence of vectors 11 11 11 00 11 11. We shall give one diagram for each moment of time $i = 0, 1, \ldots, 5$. Each diagram presents the outcome of Step 3 of the above algorithm after iteration $i + 1$. Only the costs $w(\mathbf{s}, i+1)$ and the paths $\mathcal{P}(\mathbf{s}, i+1)$ are shown. These are called the *surviving paths* — they are the only paths that the algorithm will need in the next iteration.







The last diagram shows the minimal cost path to the vertex $(00, 6)$ that is the output of the algorithm. This winning path corresponds to the codeword 11 10 10 00 01 11 $= \mathbf{x}(D) = (1 + D + D^2 + D^5, 1 + D^4 + D^5)$. The corresponding input is then $u(D) = \mathbf{x}(D)H(D) = 1+D+D^3$. We see that the winner is at Hamming distance 3 from the received vector. Thus we were able to correct 3 errors even though the free distance is only 5 (which suggests that one should only be able to correct 2 errors). This happens often when applying the Viterbi algorithm. The reason for this is that the Viterbi algorithm is a *complete decoding algorithm*, i.e. it will always gives us a codeword that is closest to the received vector. There may be several of them, if we are above the guaranteed error correction capability of the code (here 2 errors). As we just saw, for some received vectors, there are no codewords within distance ≤ 2.

It is also possible to use the trellis diagram to compute the free distance of a convolutional code. The idea is simply that we remove the edge labelled $\bar{0}/\bar{0}$ that leads from the vertex $(\bar{0}, 0)$ to the vertex $(\bar{0}, 1)$ and assign each remaining edge in the trellis a cost that is equal to the Hamming weight of the output label of the edge in question. Then we apply the Viterbi algorithm until the condition $w(\mathbf{s}, i) \geq w(\bar{0}, i)$ holds for all $\mathbf{s} \in \mathbf{F}^m$. It is then easy to see that for such $i = i_0$ the variable $w(\bar{0}, i_0)$ equals the free distance. We have excluded the all-zero path and it is obvious that for no $i > i_0$ can we find a lower cost path to vertex $(\bar{0}, i)$. After all, such a path will have to pass through one of the vertices $(\mathbf{s}, i_0)$.

It is an easy application of Theorem 3.1 to show that in the case of a canonical generator matrix the above algorithm will converge, i.e. eventually for some i the condition $w(\mathbf{s}, i) \geq w(\bar{0}, i)$ holds for all $\mathbf{s} \in \mathbf{F}^m$.

Example 3.1 Recompute the free distance of the example code of the introduction using the Viterbi algorithm.

When a causal rational generator matrix is used, we have no systematic way to terminate a codeword (no finite number of terminating zeros suffices). The Viterbi algorithm can be used, nevertheless. As we no longer know the state of the encoder at the end of the transmission (at time N), we simply select the minimal cost path $\mathcal{P}(\mathbf{s}, N), \mathbf{s} \in \mathbf{F}^m$ as the survivor.

We close this chapter with the remark that the Viterbi algorithm easily adapts to metrics other than the Hamming distance. For example in *soft decision decoding*, where reliability information about the received bits is available, squared Euclidean distance may be used instead.

Exercises

In problems 1 to 3 we study the modified state diagram of Figure 3.2 corresponding to the $(2,1)$ code with generator matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$, and study different applications of the transfer function.

3.1 *Let F_n be the number of simple codewords corresponding to paths of length $n \in \mathbf{Z}$. Show that $F_i = 0$, for $i < 3$, $F_3 = 1$ and that for all $i > 3$ we have the Fibonacci recurrence relation*

$$F_i = F_{i-1} + F_{i-2}.$$

3.2 *Let $\mathbf{x}(D) = u(D)G(D)$ be a finite weight simple codeword of this code. Show that the weight of $\mathbf{x}(D)$ depends only on the weight of the input $u(D)$.*

3.3 *Let $n > 5$ be an integer. Show that the number of such simple codewords of weight n that correspond to paths of an even length is equal to the number of such simple codewords of weight n that correspond to paths of an odd length.*

3.4 *Let us study the $(3,1)$ code generated by the canonical matrix*

$$G(D) = (\,1 + D \quad 1 + D^2 \quad 1 + D + D^2\,).$$

Draw the state diagram of the encoder gotten from the direct-form realization of $G(D)$. As a graph it is identical to that of Figure 3.1, only the labels are different. Compute the transfer function of this encoder. Show that the free distance of this code is 7 and find all the simple codewords of weight at most 9.

3.5 *In a state or a trellis diagram there are often many parallel edges connecting two vertices. In this problem we study this phenomenon. Assume that a canonical generator matrix was used in constructing the diagrams.*

A) *Show that such edges of a state diagram that loop from the zero vertex to itself are in a one-to-one correspondence with the polynomial codewords of degree 0. Hence such edges form a vector space over the field $\mathbf{F}$ (under addition of the output labels). This additive group is called the* parallel transition group.

B) *Show that the dimension p of the parallel transition group is equal to the number of such Forney indices that are equal to zero.*

C) *Let $\mathbf{s}$ and $\mathbf{s}'$ be two state vectors. Let us consider the state diagram. Show that the number of edges leading from the vertex $\mathbf{s}$ to the vertex $\mathbf{s}'$ is either zero or equal to 2^p, where p is the dimension of the parallel transition group.*

3.6 *Draw the state diagram and the trellis section of the direct-form realization of the canonical matrix*

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

Check, whether $(D + D^2 + D^3, 0, 1 + D^2 + D^4)$ is a codeword or not.

3.7 *Assume that the $(2,1)$ code and the trellis of Figure 3.3 is used. Find a codeword closest to the received vector $\mathbf{y}(D) = (1 + D + D^2 + D^3 + D^4, 1 + D + D^2 + D^4)$ with the Viterbi algorithm. What is the corresponding input $u(D)$?*

3.8 *Sometimes a received bit is so badly garbled that it is impossible to say, whether it should be interpreted as 1 or 0. In such a situation the receiver is in some applications allowed to (instead of making a guess between 0 and 1) mark such a bit as an* erasure *'?'. This is interpreted as an unknown bit at Hamming distance* $1/2$ *from both 0 and 1. So for example the Hamming distance between the vectors 0?101 and 11101 is then* $3/2$*. There is no need for the costs assigned to the edges in a trellis to be integers. Therefore the Viterbi algorithm will decode erasures as well as errors with no modifications. Find the codeword closest to the received vector 11 ?0 0? 00 10 11, when the* $(2,1,2)$ *code of Figure 3.3 is being used.*

3.9 *Assume that the code of Example 1.5 is used. Find all the codewords that are as close to the received vector 0111 0100 1000 as possible.*

3.10 *Find the trellis diagram of the* $(3,1)$*-code* C *generated by*

$$G(D) = (1 + D^2, 1 + D + D^2, 1 + D + D^2).$$

(Encoding determined by the canonical matrix $G(D)$*.) With the aid of the Viterbi algorithm find all the words of* C *that are at the lowest possible Hamming distance from the received vector* $\mathbf{y} =$ *111 010 110 011.*

3.11 *Find the trellis diagram of the* $(3,1,2)$*-code* C *generated by*

$$G(D) = (1 + D, 1 + D^2, 1 + D + D^2)$$

(Encoding based on the canonical generator matrix $G(D)$*) Compute the free distance of* C *with the aid of the Viterbi algorithm.*

3.12 *An* $(n,1,m)$ *convolutional code* C *is called* antipodal*, if the entries* $g_i(D), i = 1, \ldots, n$ *of a canonical generator matrix* $G(D)$ *all have the following properties:* $g_i(0) = 1$ *(i.e. the constant term is 1) and* $\deg g_i(D) = m$ *(i.e. the highest degree* D^m*-term has coefficient 1). In a trellis diagram gotten from* $G(D)$ *there are exactly two out-going edges leaving each vertex and exactly two incoming edges entering each vertex. Show that, if* C *is antipodal, then the output labels of the two edges leaving a given vertex differ at all the bit positions. Similarly show that the output labels of the two edges entering a given vertex also differ at all positions. Hint: One way is to think about a physical encoder.*

Chapter 4

Good convolutional codes and some constructions

A most striking fact is the lack of algebraic constructions of families of convolutional codes. The contrast to the theory of block codes is very sharp in this respect. A convolutional (n, k, m) code is called *distance optimal*, if it has the largest free distance among the codes with these parameters. As the parameters n and k are usually quite small, exhaustive search has yielded tables of distance optimal convolutional codes. We have included such tables to section 4.1. In this chapter we also study two ways of constructing new convolutional codes from known codes. The concept of a *dual code* is quite natural and allows us to define convolutional codes by means of a parity check matrix. We also take a look at the process of *puncturing* convolutional codes. This is quite a useful way of increasing the information rate of a convolutional code.

4.1 Tables of distance optimal convolutional codes

In addition to giving the tables let us study a method of deriving bounds to the free distance of an (n, k) code C. The bound will depend on the Forney indices $e_1 \leq e_2 \leq \cdots \leq e_k$ of C. Let us study the subspace C_ℓ of C consisting of polynomial codewords of degree at most ℓ, where ℓ is some natural number. The space C_ℓ can be viewed as a block code of length $n(\ell + 1)$ in an obvious manner. The next proposition tells us the dimension of this code.

Proposition 4.1 *The dimension d_ℓ of the space of polynomial codewords of a bounded degree $\leq \ell$ in an (n, k) code with Forney indices $e_1 \leq e_2 \leq \cdots \leq e_k$ and canonical generator matrix $G(D)$ is*

$$d_\ell = \sum_{i=1}^{k} \max\{0, \ell + 1 - e_i\}.$$

Proof. This is an immediate consequence of the linear independence of the rows of $G(D)$ over $\mathbf{F}[D]$ and of the "polynomial output implies polynomial input" and the "predictable degree" properties of canonical generator matrices. ■

Corollary 4.1 *The dimension d_ℓ of the space of polynomial codewords of a bounded degree $\leq \ell$ in an (n, k, m) code is bounded by*

$$d_\ell \geq k(\ell + 1) - m.$$

Here we have an equality, when ℓ is large enough (greater than or equal to the largest Forney index).

If the code C we started with has free distance d, then obviously the minimum Hamming distance of any of the codes C_ℓ is at least d. This allows us to exclude the existence of certain (n, k, m, d) codes and to prove the distance optimality of some codes. To that end we must have some information about the existence of binary linear block codes. The following well-known bound is particularly useful in proving the non-existence of certain low-dimensional block codes.

Theorem 4.1 Griesmer Bound *If a binary linear code of length n, dimension k and minimum distance d exists, then*

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil .$$

Proof. Omitted. See for example MacWilliams–Sloane, *The Theory of Error-Correcting Codes*, page 546. ∎

Example 4.1 Show that a $(2, 1, 2)$ code (resp. a $(2, 1, 3)$ code) cannot have free distance larger than 5 (resp. 6).

The next example shows that the bounds one can get from proposition 4.1 are not the whole story.

Example 4.2 Show that proposition 4.1 cannot give a bound tighter than 4 to the free distance of a $(2, 1, 1)$ code but yet no convolutional $(2, 1, 1)$ code can have a free distance higher than 3.

In the tables that follow we use the so called octal presentation of polynomials. A polynomial $p(D)$ with binary coefficients can be efficiently packed (in the interest of saving space in the tables) as follows: Interpret $p(D)$ as a polynomial having integer coefficients. Then write the integer $p(2)$ in octal (base 8) form. For example $p(D) = 1 + D^2$ has octal presentation 5_8 (=5 decimal) and $p(D) = 1 + D + D^5$ has octal presentation 43_8 (=35 decimal). In the tables we omit the subscript 8.

Table 4.1 — some optimal $(2, 1, m)$ codes

m	d	canonical $G(D)$
0	2	(1 1)
1	3	(1 3)
2	5	(5 7)
3	6	(13 17)
4	7	(23 35)
5	8	(53 75)
6	10	(133 171)
8	12	(561 753)
10	14	(2335 3661)

Table 4.2 — some optimal $(3, 1, m)$ codes

m	d	canonical $G(D)$
0	3	(1 1 1)
1	5	(1 3 3)
2	8	(5 7 7)
3	10	(13 15 17)
4	12	(25 33 37)
5	13	(47 53 75)
6	15	(133 145 175)
7	16	(225 331 367)
8	18	(557 663 711)

Table 4.3 — some optimal $(3, 2, m)$ codes

m	d	canonical $G(D)$
0	2	$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$
2	3	$\begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \end{pmatrix}$
3	4	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 1 & 7 \end{pmatrix}$
4	5	$\begin{pmatrix} 7 & 4 & 1 \\ 2 & 5 & 7 \end{pmatrix}$
5	6	$\begin{pmatrix} 3 & 6 & 7 \\ 14 & 1 & 17 \end{pmatrix}$
6	7	$\begin{pmatrix} 13 & 6 & 13 \\ 6 & 13 & 17 \end{pmatrix}$
7	8	$\begin{pmatrix} 3 & 6 & 15 \\ 34 & 31 & 17 \end{pmatrix}$

Table 4.4 — some optimal $(4, 1, m)$ codes

m	d	canonical $G(D)$
0	4	(1 1 1 1)
1	7	(1 3 3 3)
2	10	(5 7 7 7)
3	13	(13 15 15 17)
4	16	(25 27 33 37)
5	18	(53 67 71 75)
6	20	(135 135 147 163)
7	22	(235 275 313 357)
8	24	(463 535 733 745)

4.2 The dual code and Wyner–Ash codes

For any two vectors $\mathbf{x}(D) = (x_1(D), \ldots, x_n(D))$ and $\mathbf{y}(D) = (y_1(D), \ldots, y_n(D))$ in $\mathbf{F}^n((D))$ we use the notation

$$\mathbf{x} \cdot \mathbf{y} = x_1(D)y_1(D) + \cdots + x_n(D)y_n(D)$$

(even though this is not an "inner product"). Given an (n, k) convolutional code C with canonical generator matrix $G(D)$ we call

$$C^\perp = \{\mathbf{y} \in \mathbf{F}^n((D)) \mid \mathbf{y} \cdot \mathbf{x} = 0 \text{ for all } x \in C\}$$

the *dual code* of C. It is easy to see that $C^\perp$ is, indeed, an $(n, n-k)$ convolutional code: standard arguments from linear algebra show that its dimension as a vector space over $\mathbf{F}((D))$ is $n-k$ and that it has a basis consisting of vectors $\in \mathbf{F}^n(D)$. We immediately see that $\mathbf{y}(D) \in \mathbf{F}^n((D))$ is a codeword of $C^\perp$ if and only if

$$\mathbf{y}(D)G(D)^T = \bar{0}.$$

Thus we may call $G(D)$ a *parity-check matrix* for $C^\perp$.

By Theorem 1.2 the matrix $G(D)$ can be completed to a unimodular matrix Q by adding a suitable $(n-k) \times n$ block L

$$Q = \begin{pmatrix} G \\ L \end{pmatrix}.$$

Let us then write the matrix Q^{-1} in a block form

$$Q^{-1} = P = \begin{pmatrix} U^T & V^T \end{pmatrix},$$

where U is a polynomial $k \times n$ matrix and V is a polynomial $(n-k) \times n$ matrix. As P is of full rank, the rows of V are linearly independent over $\mathbf{F}((D))$. Because P is the inverse matrix of Q the rows of V are orthogonal to the rows of $G(D)$. Hence V is a polynomial generator matrix for $C^\perp$. On the other hand the matrix $P^T = \binom{U}{V}$ is unimodular. Thus adding the rows of U to the matrix V turns it into a unimodular matrix. By Theorem 1.2, the matrix V is basic.

It can be shown (a result from linear algebra but beyond the scope of our course 'Linear Algebra') that an $(n-k) \times (n-k)$ minor of V is equal to the $k \times k$-minor of G gotten by including the columns of G that were excluded from V. Thus the internal degrees of V and G are equal. The internal degree of a basic generator matrix is equal to the degree of a code, so we have the following result due to Forney.

Theorem 4.2 $\deg C = \deg C^\perp$.

Example 4.3 Let $G(D)$ be a canonical generator matrix of an $(n, n-1)$ code C. Let $\Delta_i(D)$ be the minor gotten from $G(D)$ by excluding the ith column. Then

$$(\Delta_1(D), \Delta_2(D), \ldots, \Delta_n(D))$$

generates the dual code $C^\perp$.

Example 4.4 Let $A = (a_{ij})$ be a unimodular 3×3-matrix. By a direct computation verify the "minors of the inverse" result quoted in the proof of Theorem 4.2.

Next we introduce the only known families of distance optimal codes. Consider the following two parity-check matrices

$$H_m = \left(1 \quad 1+D \quad 1+D^2 \quad 1+D+D^2 \quad \cdots \quad 1+D+\cdots+D^m \right),$$

where $m \geq 2$ and the entries are the 2^m polynomials of degree at most m that have a constant term equal to 1, and

$$H'_m = \left(1+D^m \quad 1+D+D^m \quad 1+D^2+D^m \quad \cdots \quad 1+D+\cdots+D^m \right),$$

where $m \geq 3$ and the entries are the 2^{m-1} polynomials of degree exactly m that have a constant term equal to 1.

Theorem 4.2 tells us that the codes defined by these parity-check matrices both have degree m. The $(2^m, 2^m - 1, m)$ code defined by the parity-check H_m is called the mth order Wyner–Ash code in honor of its discoverers. Similarly the $(2^{m-1}, 2^{m-1} - 1, m)$ code defined by the parity-check matrix H'_m is called the *extended Wyner–Ash code of order m.*

Proposition 4.2 *The free distance of the Wyner–Ash code (resp. the extended Wyner–Ash code) is three (resp. four).*

Proof. Obviously from the relations

$$(1+D, 1) \cdot (1+D, 1+D^2) = 0$$

and

$$(1+D^m) + (1+D+D^m) + (1+D^2+D^m) + (1+D+D^2+D^m) = 0$$

it follows that there exists codewords of the prescribed weights.

As neither H_m nor H'_m has a zero entry neither code has words of weight 1. Similarly since there are no relations of the type

$$D^r p_1(D) + D^s p_2(D) = 0,$$

where r and s are integers and $p_1(D), p_2(D)$ are entries in the parity-check matrix, no words of weight 2 exist either. This proves that the free distance of Wyner–Ash codes is exactly three.

Let $\mathbf{x}(D)$ be a polynomial word of minimal weight in an extended Wyner–Ash code. Without loss of generality we may assume that the constant term $\mathbf{x}(0) \neq 0$. By studying the constant term of $\mathbf{x}(D) \cdot H'_m$ we conclude that $\mathbf{x}(0)$ must have an even weight. Similarly we see that the number of highest degree terms in $\mathbf{x}(D)$ must also be even. In order for $\mathbf{x}(D)$ to have a weight less than four it is therefore necessary that $\mathbf{x}(D)$ is of degree zero. But then the weight of $\mathbf{x}(D)$ must be an even number, so it cannot have weight three. The other low weights were excluded earlier, so the claim follows. ∎

Example 4.5 Find a canonical generator matrix for the $(4, 3, 2, 3)$ Wyner–Ash code and for the $(4, 3, 3, 4)$ extended Wyner–Ash code. Show that their respective Forney indices are $(0, 1, 1)$ and $(0, 1, 2)$.

Proposition 4.3 *The extended Wyner–Ash codes are distance optimal.*

Proof. Let C be any $(2^{m-1}, 2^{m-1}-1, m)$ code. By Proposition 4.1, the dimension of the linear block code C_0 must be at least $2^{m-1}-1-m$. Assume that the free distance of C is $d \geq 5$. Thus C_0 is a linear code of length 2^{m-1}, dimension at least $2^{m-1}-1-m$ and minimum distance at least 5. Let H be the parity-check matrix for the code C_0. There are 2^{m-1} columns and at most $m+1$ rows in the matrix H. Let $H^{(i)}, i = 1, \ldots, 2^{m-1}$ be the column vectors of H. The sums $H^{(i)} + H^{(j)}$ must all be distinct for different pairs of indices $i < j$, for otherwise there is a codeword of weight at most 4. However, the number of such sums is $\binom{2^{m-1}}{2} = 2^{m-2}(2^{m-1}-1)$. For $m \geq 5$ this is higher than the number of vectors in the column space of H, which is a contradiction. In the case $m = 3$ (resp. $m = 4$) the block code C_1 (resp. the block code C_0) is a binary linear $[8, 3]$ code. It is easy to see that the minimum distance of such a code cannot > 4. ∎

We state the following result. A proof consists of a study of certain low weight linear combinations of the entries of a parity-check matrix for the codes in question and is left as a challenging (but not at all too difficult) exercise.

Theorem 4.3 (Ytrehus) *Any $(n, n-1, m)$ code with free distance at least three must have $n \leq 2^m$ and any $(n, n-1, m)$ code with free distance at least four must have $n \leq 2^{m-1}$.*

As an immediate corollary to Ytrehus' Theorem we see that the Wyner–Ash codes are also distance optimal – after all the non-existence of $(2^m, 2^m-1, m, d)$ codes for $d \geq 4$ is included in the theorem.

Example 4.6 Show that all $(2, 1)$ codes and our favorite $(4, 2)$ code of Example 1.5 are 'self-dual up to a permutation of the components of the output'.

4.3 Blocking and puncturing convolutional codes

The information rate of an ordinary block code can be increased by simply throwing away one or more bits from each codeword. Usually the minimum distance of the code suffers from this process, but for a long code, this may not be too bad. In the case of a convolutional code it would be too crude a process to simply discard one of the components of the output signal. The adopted practice is to first artificially increase the "length" n by the so called blocking that amounts to 'running the system clock at a fractional speed' and only then *puncturing* one or more components of the output. We shall give an algebraic description of the steps involved. An interesting observation is that the resulting punctured codes can be decoded with the (simpler) trellis diagram of the original (simpler) code.

Let $M \geq 2$ be an integer and let $E = D^M$. The field of formal Laurent series $\mathbf{F}((D))$ can be viewed as an M-dimensional vector space over the subfield $\mathbf{F}((E))$ that is obviously also a field of formal Laurent series. The powers $1 = D^0, D, D^2, \ldots, D^{M-1}$ obviously form a basis of $\mathbf{F}((D))$ over $\mathbf{F}((E))$. Similarly the space $\mathbf{F}^n((D))$ can be viewed as an nM-dimensional $\mathbf{F}((E))$-space. In this case a natural basis would consist of such n-tuples e^i_j that have a single non-zero component $D^i, i = 0, 1, \ldots, M-1$ at position $j = 1, 2, \ldots, n$ and zeros elsewhere. When we express an arbitrary vector $\mathbf{x}(D)$ in $\mathbf{F}^n((D))$ as an $\mathbf{F}((E))$-linear combination of the vectors e^i_j, we immediately see that the weight of $\mathbf{x}(D)$ does not depend on the point of view. For example the vector

$$(1 + D^2 + D^3 \;\; 1 + D + D^2) = (1 + E)e^0_1 + Ee^1_1 + (1 + E)e^0_2 + e^1_2$$

is of weight 6 as an element of $\mathbf{F}^2((D))$ and as an element of $\mathbf{F}^4((E))$.

For an arbitrary (n,k,m,d) code C we define the *Mth blocking* $C^{[M]}$ of C to be the (nM,kM) code gotten by viewing C as a vector space over the subfield $\mathbf{F}((E))$, where the coordinates will be computed with respect to the basis $\{e_j^i \mid j=1,\ldots,n,\ i=0,1,\ldots,M-1\}$ of $\mathbf{F}^n((D))$.

By the above discussion, the free distance of $C^{[M]}$ is also d. Furthermore the $\mathbf{F}$-spaces of causal codewords of C and $C^{[M]}$ are obviously equal. By the state space theorem, the degree of the code $C^{[M]}$ is then also m, so $C^{[M]}$ is an (nM,kM,m,d) code.

The problem of finding a canonical generator matrix for a blocking remains. A canonical generator matrix $G(D)$ for C simply gives us an $\mathbf{F}((D))$-linear mapping L_G from $\mathbf{F}^k((D))$ to $\mathbf{F}^n((D))$ with respect to the standard bases: $L_G(\mathbf{u}(D))=\mathbf{u}(D)G(D)$. This $\mathbf{F}((D))$-linear mapping is obviously also $\mathbf{F}((E))$-linear. So to get a generator matrix for the blocking $C^{[M]}$ we simply compute the matrix of the mapping L_G with respect to the new standard bases.

To that end we first study the simple case $n=k=1$, where L_G is simply multiplication by a polynomial $p(D)=g_{11}(D)$. In this case the standard basis consists of the powers $1,D,D^2,\ldots,D^{M-1}$ and multiplication by $p(D)$ corresponds to an $M\times M$ matrix $p^{[M]}(E)$ called *the polycyclic pseudocirculant* of $p(D)$. To find $p^{[M]}(E)$ we first regroup the terms in $p(D)$ according to the remainder of the degree modulo M, i.e. we write $p(D)$ as an $\mathbf{F}[E]$-linear combination of the monomials in the basis

$$p(D)=p_0(D^M)+p_1(D^M)D+p_2(D^M)D^2+\cdots+p_{M-1}(D^M)D^{M-1}=\sum_{i=0}^{M-1}p_i(E)D^i.$$

The polynomials $p_i(E)$ are called the Mth *polyphase components* of $p(D)$.

We can similarly divide input and output signals into their respective polyphase components. The matrix $p^{[M]}(E)$ that we want to find should give the dependence of the polyphase components of the output $u(D)p(D)$ on the polyphase components of the input $u(D)$.

From the polyphase decomposition of $p(D)$ we immediately see that for any $j, 0\le j\le M-1$

$$D^jp(D)=\sum_{i=0}^{M-1}D^{i+j}p_i(E)=\sum_{i=0}^{j-1}D^iEp_{i+M-j}(E)+\sum_{i=j}^{M-1}D^ip_{i-j}(E).$$

The extra factor $E=D^M$ comes from the fact that in the power D^{i+j} the exponent may "overflow" and become larger than $M-1$. Hence the matrix $p^{[M]}(E)=(p_{ij}(E))$ has entries

$$p_{ij}(E)=\begin{cases} p_{j-i}(E), & \text{if } j\ge i \text{ and}\\ Ep_{M+j-i}(E), & \text{if } j<i.\end{cases}$$

Observe that contrary to the practice in our course 'Linear Algebra' the matrix of a linear mapping is written on the right hand side. This is necessary, when we use *row* vector notation. We also observe that the degree of the ith row of the matrix $p^{[M]}(E)$ is $\left\lfloor\frac{e+i-1}{M}\right\rfloor$, where $e=\deg p(D)$.

The general case now follows easily. We only need to keep track of the order of the basis vectors e_j^i. Let us fix the ordering so that the first basis vectors are the monomials D^i at position 1, i.e. $e_1^i, i=0,1,\ldots,M-1$, followed by the vectors e_2^i in their natural order and so on. With this convention the matrix of L_G with respect to the bases e_j^i is gotten from the matrix $G(D)$ by simply replacing each entry $g_{ij}(D)$ with the $M \times M$ block $g_{ij}^{[M]}(E)$. Thus we have the following result:

Theorem 4.4 (Hole) *Let $G(D) = (g_{ij}(D))$ be a polynomial generator matrix for an (n,k) code C. Then the Mth blocking $C^{[M]}$ is generated by the matrix*

$$G^{[M]}(E) = \begin{pmatrix} g_{11}^{[M]}(E) & g_{12}^{[M]}(E) & \cdots & g_{1n}^{[M]}(E) \\ g_{21}^{[M]}(E) & g_{22}^{[M]}(E) & \cdots & g_{2n}^{[M]}(E) \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1}^{[M]}(E) & g_{k2}^{[M]}(E) & \cdots & g_{kn}^{[M]}(E) \end{pmatrix}.$$

To summarize: we have simply replaced the original delay operator D by its suitable power E. So effectively the transition from C to $C^{[M]}$ is obtained by letting the system clock run at a fractional speed, i.e. we wait for M times as many input bits as while using C and also output M times as many output bits.

Theorem 4.5 *Let $G(D)$ be a canonical generator matrix for an (n,k,m) code C and let the Forney indices of C be $e_1 \leq e_2 \leq \cdots \leq e_k$. Then the matrix $G^{[M]}(E)$ is a canonical generator matrix for $C^{[M]}$ and the Forney indices of $C^{[M]}$ are $e_{i,j} = \left\lfloor \frac{e_i + j}{M} \right\rfloor$, where $j = 0, 1, 2, \ldots, M-1$ and $i = 1, 2, \ldots, k$.*

Proof. The statement about the Forney indices follows immediately from our earlier observation about the row degrees of the polycyclic pseudocirculants, if we can show that $G^{[M]}(E)$ is canonical. However, an easy computation shows that the external degree of $G^{[M]}$ is equal to the external degree of $G(D)$. Since $\operatorname{extdeg} G^{[M]}(E) = \operatorname{extdeg} G(D) = m$ and this is equal to the degree of the code $C^{[M]}$, the matrix $G^{[M]}$ has the lowest possible external degree and hence is canonical. ∎

Example 4.7 Find canonical generator matrices for the second and the third blockings of the distance optimal $(2,1,3)$ code C of Table 4.1. Show that $C^{[3]}$ is a distance optimal $(6,3,3)$ code.

From a blocking of a code C some components of the output vectors can then be thrown away. This so called *puncturing* amounts to removing some columns from the generator matrix of $C^{[M]}$. The columns to be deleted are usually given by the so called puncturing pattern $P = (p_{ij})$ — a binary $n \times M$ matrix, where the entry p_{ij} is 0, if the column corresponding to the basis element e_i^{j-1} is removed from the generator matrix and 1 otherwise. If $G(D)$ is a generator matrix for C, we denote by $G_P(D)$ the matrix gotten by puncturing G according to the pattern P.

We have no way of knowing, when the nice properties of a generator matrix G are shared by the punctured matrix G_P. Sometimes a canonical generator matrix remains canonical, sometimes a generator matrix becomes *catastrophic* (see the exercises for this concept). Usually puncturing decreases the free distance of the code and may sometimes also decrease the degree of the code.

Example 4.8 Let C be the distance optimal $(2,1,3)$ code of Example 4.7 and $G^{[2]}$ the canonical generator matrix for its second blocking $C^{[2]}$. Show that removing one of the

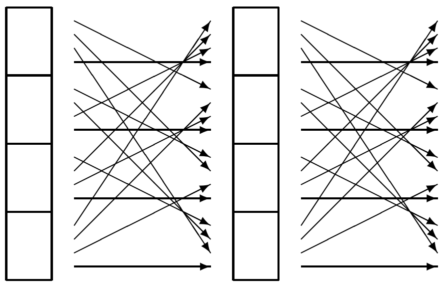


Figure 4.1: A $(3, 2, 2)$ trellis

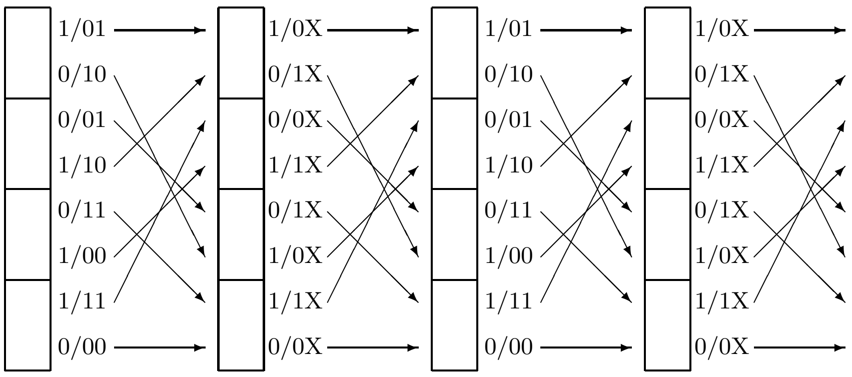


Figure 4.2: A $(2, 1, 2)$ trellis for a punctured $(3, 2, 2)$ code

columns e_1^i leads to a non-basic (catastrophic) generator matrix, but removing the column of e_2^1, i.e. puncturing with the pattern

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

yields a canonical generator matrix for a distance optimal $(3, 2, 3, 4)$ code.

In addition to being useful in the search for good codes, the technique of puncturing gives significant savings in the complexity of the Viterbi decoding algorithm. The savings come from the fact that one may decode a blocking with the trellis of the original code — simply split the output/received vector into M parts. The same observation is valid for a punctured code as well — simply omit the corresponding output bits from every Mth section of the trellis. These are the bits marked 'X' in Figure 4.2.

In Viterbi decoding the number of additions is equal to the number of edges in a trellis. If we use a canonical trellis of an (nM, kM, m) code, there are 2^{kM} edges leaving each of the 2^m states — a total of 2^{m+kM} edges per kM input bits. If we use M sections of a canonical trellis of the original (n, k, m) code, there are 2^{k+m} edges in a trellis section. So in the latter case we only need to add the costs of $M2^{k+m}$ edges. (As the trellis of an (nM, kM, m) code often contains many parallel edges, some additions could be replaced by comparisons. However, for many CPUs, like Intel x86, a comparison is not at all faster than an addition.) The number of comparisons that must be made in the Viterbi algorithm is also cut down, when M sections of an (n, k, m) trellis are used instead of a section of an (nM, kM, m) trellis.

Exercises

4.1 *Using Proposition 4.1 show that the free distance of a $(2, 1, 4)$ code cannot be > 8. Proposition 4.1 cannot be used to improve this result. Prove the non-existence of a convolutional $(2, 1, 4)$ code of free distance 8 by excluding all the possible candidates $G(D)$ for a canonical generator matrix. A useful observation is that, if a polynomial has an even weight, it is divisible by $1 + D$. Testing the candidates with polynomial inputs of degree ≤ 2 should suffice.*

4.2 *With the aid of Proposition 4.1 show that a $(3, 2, 1)$ code cannot have free distance larger than 3. By studying candidate generator matrices, show that this bound cannot be achieved and that hence $(3, 2, 1)$ codes cannot have any better distance properties than a carefully chosen $(3, 2, 0)$ code. Hint: This is a lot easier than the previous problem.*

4.3 *Let $H = (h_1(D), \ldots, h_n(D))$ be a polynomial parity-check matrix for an $(n, n-1, m)$ code C, where $m = \max_i \deg h_i(D)$. Let r be the dimension of the $\mathbf{F}$-space V spanned by the polynomials $h_i(D)$ in the ring $\mathbf{F}[D]$.*

A) Form an $\mathbf{F}$-linear map ϕ from $\mathbf{F}^n$ to the space V such that the code C_0 consisting of constant codewords in C is equal to the kernel of ϕ. Show that $\dim C_0 = n - r$.

B) Show that the number of non-zero Forney indices of C is $r - 1$. Hint: The number of Forney indices equal to zero was computed in problem 5 of set 5.

C) Show that all the Forney indices of the Wyner–Ash code are either 0 or 1.

D) Show that all the Forney indices of the extended Wyner–Ash code are either 0,1 or 2.

4.4 *Let C be a convolutional code. Show that for all the codewords $\mathbf{x}(D) = \sum_i \mathbf{x}_i D^i$ of C all the components $\mathbf{x}_i$ have an even weight, iff the vector $(1, 1, 1, \ldots, 1)$ belongs to the dual code $C^\perp$.*

4.5 *Let C be the distance optimal $(3, 1, 3)$ code of Table 4.2. Find a canonical generator matrix for the $(3, 2, 3)$ code $C^\perp$. Show that the free distance of $C^\perp$ is 4 (and hence that $C^\perp$ is also distance optimal according to Table 4.3). Hint: Proving the free distance is probably easiest, if you study the parity-check equation. For example the existence of a codeword of weight 3 implies an equation $D^r h_1(D) + D^s h_2(D) + D^t h_3(D) = 0$, where $h_i(D)$ are (not necessarily distinct) entries in the parity-check matrix.*

4.6

A)If $p(D)$ is a polynomial of degree e show that the degree of the ith row of the matrix $p^{[M]}(E)$ is $\left\lfloor \frac{e+i}{M} \right\rfloor$.

B) Show that for all integers $e, M > 0$ $e = \sum_{i=0}^{M-1} \left\lfloor \frac{e+i}{M} \right\rfloor$.

4.7 *Find the canonical generator matrix $G^{[2]}$ for the second blocking of the $(2,1,2,5)$ code generated by $G(D) = (1+D^2, 1+D+D^2)$. Compare the result to the answer of problem 1, set 3. Which matrices gotten by removing a single column from $G^{[2]}(E)$ are canonical generator matrices for $(3,2)$ codes? Show also that the $(4,2,2,5)$ code $C^{[2]}$ is distance optimal.*

4.8 *Show that, if $G(D)$ is a basic generator matrix for a convolutional code, then $G^{[M]}(E)$, $E = D^M$ is also basic. Hint: What could be a polynomial right inverse for $G^{[M]}(E)$?*

4.9 *Let C be the distance optimal $(2,1,3)$ code of Table 4.1. From the Griesmer bound it follows, that no $[9,3,d]$ codes for $d \geq 5$ and no $[12,3,d]$ codes for $d \geq 7$ exist.*

A)Show that $C^{[2]}$ is a distance optimal $(4,2,3)$ code.

B)Show that puncturing C according to the pattern

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

leads to a canonical generator matrix for a $(3,2,3)$ code C_P. Find a parity-check matrix for C_P (see Example 4.3) and show that the free distance of C_P is 4. Show that C_P is distance optimal.

4.10 *Show that for a $(2,1,m,d)$ code we always have $d \leq \left\lfloor \frac{4(m+2)}{3} \right\rfloor$. Which codes of Table 4.1 meet this bound?*

4.11

A) Find a canonical generator matrix for an $(8,2,2,10)$ convolutional code C. Show that no $(8,2,2,d)$ codes for $d > 10$ exist.

B) Show that the $(4,1,1,7)$ code of the table is distance optimal, i.e. no $(4,1,1,d)$ codes for $d > 7$ exist.

4.12 *Let C be the $(3,1,3,10)$-code from Table 4.2. Show that the vector $(1, D, 1+D)$ belongs to the dual code $C^\perp$. Find a canonical generator matrix for $C^\perp$. With the aid of the parity-check equation show that the free distance of $C^\perp$ is four.*

4.13 *Construct a canonical generator matrix for a $(6,2,3,10)$ code. Show that such a code is distance optimal, i.e. no $(6,2,3,d)$ codes with $d > 10$ exist.*

4.14 *Let us puncture the $(2,1,3)$ code from the tables. Which puncturing patterns P lead to a canonical generator matrix G_P for a $(3,2,3)$ code?*

4.15 *Let $G(D) = (1+D^2, 1+D^2, 1+D+D^2, 1+D+D^2, 1+D+D^2)$ be a (obviously canonical) generator matrix for a $(5,1,2)$ code C. Show that C is an optimal code with free distance 13. Find a $(10,2,2,13)$ code.*

Chapter 5

Modern Variations

In this chapter I will give a cursory description of two very powerful classes of error-correcting codes, Turbo codes and LDPC codes. Turbo coding was presented by Berrou, Glavieux and Thitimajshima in 1993, and it was the first class of error-correcting codes that allowed one to communicate at a rate close to the Shannon bound. For example the 4G (LTE) cellular phone system relies on turbo codes when transmitting bulk data between base station and user equipment (e.g. a cell phone). Low Density Parity Check codes were first studied by Gallager as early as in the 1960s. However, the potential of the underlying ideas was forgotten for a long time (my understanding is that the theory was somewhat underdeveloped, also real-time application of LDPC-codes places heavy constraints on the computational speed of the hardware and that was probably not available in the 60s). LDPC-codes will be used in for example the European second generation digital video broadcast standard (the upgrade to the current digital television system used in e.g. Finland).

Both classes of codes use reliability information about individual bits at the receiving end. These are simply probabilities of the value of the bit b, i.e. the pair of numbers $P(b=0)$ and $P(b=1)$. More often than not the probabilities are conditional. If C is a known value of a (set of) random variable, we use the conditional probabilities $P(b=0|C)$, $P(b=1|C)$. The two probabilities must add up to one, so it is wasteful to use two variables. A convenient of packing this information into a single variable is the so called *log-likelihood ratio*, $LLR(b)$, defined as

$$LLR(b|C) = \ln \frac{P(b=0|C)}{P(b=1|C)}.$$

The interpretation of this is immediate. The number $LLR(b)$ is positive, iff $P(b=0) > P(b=1)$, i.e. if b is more likely to be 0 than 1. On the other hand if $LLR(b) < 0$, then b is more likely to be equal to 1. If $P(b=0) = P(b=1) = 1/2$, then $LLR(b) = 0$ indicating that we don't really know anything about the value of the bit b. The absolute value $|LLR(b)|$ thus quantifies the level of certainty about the value of b. As extremal cases we see that $LLR(b) = \infty$, when b is known to be $= 0$ with absolute certainty. Similarly $LLR(b) = -\infty$ indicates absolute certainty of $b = 1$.

The decoding algorithms used for both turbo codes and LDPC-codes combine information about the value of a bit collected from different sources. If (sometimes this is a BIG IF) two sources of information C_1 and C_2 are independent in the sense that the events $P(b=i, C_1)$ and $P(b=i, C_2)$ are independent for $i=0$ or $i=1$, then we can easily combine the corresponding LLRs as follows

$$LLR(b|C_1, C_2) = LLR(b|C_1) + LLR(b|C_2).$$

Independence of this kind is often only approximate, but we ignore this problem in what follows. We simply record the fact that a properly designed turbo code or an LDPC-code will achieve approximate independence, when the code blocks are long enough. Extensive computer simulations are needed to verify that a candidate code works as well as it should.

What kind of independent sources of information about the value of a bit can we have? There will always be *intrinsic information* coming from the communication channel. The bit was transmitted as a waveform or a burned into a CD-rom, magnetic tape or whatever. Some distortion may have happened, and by modelling that distortion we can interpret the read/received data about the uncoded bit. Then we, of course, have *extrinsic information* tying the value of any given bit to that of other bits. After all, we know with absolute certainty that the combination of transmitted bits forms a valid codeword. The key novelty present in both turbo codes and LDPC-codes is that there will be several sources of extrinsic information. We shall see that in a way a code is broken into smaller parts in such a way that we can carry out partial decoding on those parts. Furthermore, we have available decoding algorithms that give as outputs reliability information about individual bits — not just something 'crude' like the closest codeword. We can then feed extrinsic information gotten by decoding partial words as inputs in other parts. Furthermore, we can iterate this process, and do another round of decoding on an already used part of the code with a view of using extrinsic information from other sources and get more accurate output from that part. After all, in subsequent rounds the decoders handling partial codewords are ostensibly doing their work on more accurate data, and we are thus entitled to expect better output.

The mathematical tools needed to understand these processes are very different from those traditionally used in the study of error-correcting codes in Turku. I only have time to scratch the surface and give crude outlines of the ideas and algorithms. Also, I am unfamiliar with the necessary mathematics of stochastic processes myself.

5.1 Parity check matrices and Tanner graphs

Recall that a (binary linear) block code $\mathcal{C}$ of length n and dimension k can be defined by giving its check matrix H. It is a matrix with n columns, at least $n-k$ columns (we may occasionally use extra check equations that are linear combinations of other check equation) and entries from F. A vector $\mathbf{x} \in F^n$ is a word of the code $\mathcal{C}$, if and only if it satisfies the matrix equation $H\mathbf{x}^T = 0$. In other words a codeword must be orthogonal to all the rows of the parity check matrix H.

Example 5.1 *A binary linear code of length* 10 *and dimension* 6 *is defined by the check matrix*

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

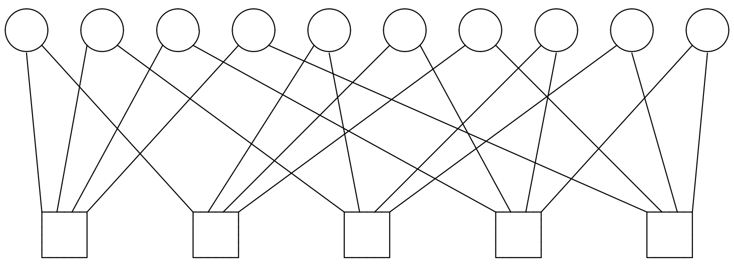


Figure 5.1: The Tanner graph of the check matrix of Example 5.1.

This check matrix has five rows, but we observe that the sum of all rows is the all zero vector. Hence the code defined by this matrix is 6-dimensional. Because all the columns are distinct, it follows easily that the code has minimum Hamming distance three.

A parity check matrix gives rise to a so called *Tanner graph.* The Tanner graph related to a check matrix is a bipartite graph. It has n variable nodes, one for each bit, (drawn as circles below). The Tanner graph also has so called check nodes, one for each row of the matrix H. The variable node number j is connected to the check node number i in the graph by an edge of the graph, iff $H_{ij} = 1$. The bipartite property means that there are no edges connecting two check nodes to each other nor any edges connecting two variable nodes to each other.

Example 5.2 *The Tanner graph related to the check matrix H of Example 5.1 is shown in Figure 5.1. Because there are two ones on each column of H, each variable node is connected to exactly two check nodes. Because there are four ones on each row of H, each check node is connected to exactly four variable nodes.*

The Tanner graph is primarily used to describe so called message passing algorithms. These algorithms are iterative. Each iteration is split into two halves. During the first half iteration the variable nodes are sending messages to all the check nodes they are connected with about their internal state. During the second half iteration the check nodes reply with messages to all the variable nodes connected to them. The details of the messages vary according to what the algorithm is doing.

The case when a message passing algorithm is used to filling *erasures* is the simplest. Assume that a word of the code $\mathcal{C}$ is being transmitted. Assume further that the codeword goes through a so called binary erasure channel. This means that each and every bit is either received correctly, or (with a probability p) is received as a totally illegible erasure - an unknown bit of value ? that is equally likely to be 0 or 1. In other words after going through a binary erasure channel the LLR of each bit is either 0 or $\pm\infty$.

In this case during the first half-iterations the variable nodes always tell all the check nodes connected to them their current information about their value, 0/?/1. For their part the check nodes carry out the following calculations. Assume that the bits $b_{i_1}, b_{i_2}, \ldots, b_{i_k}$ are connected to check node c_r. Then we know that in the the field F the linear equation

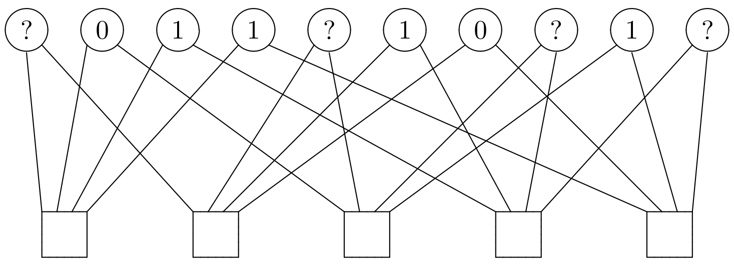
$$b_{i_1} + b_{i_2} + \cdots + b_{i_k} = 0$$

is true. So for each t, $1 \leq t \leq k$, the check node c_r attempts to solve b_{i_t} from this equation. The attempt is successful, if and only if all the other bits $b_{i_\ell}, \ell \neq t$ sent in a definite value. In that case the the check node c_r sends the solution of this equation to the variable node number i_t. If the attempt to solve the value of b_{i_t} was unsuccessful, i.e. if at least one other bit involved in this check equation was still unknown, then the message from c_r back to variable node i_t is also a ?.

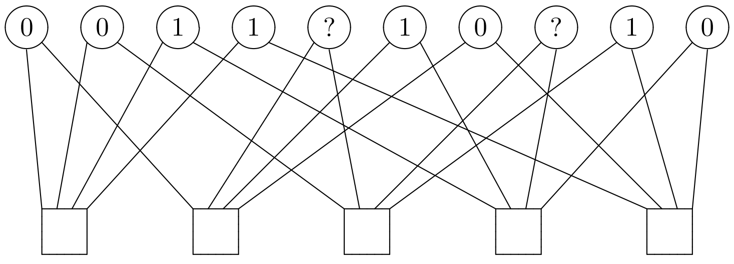
Observe that if all the bits involved in the check equation at c_r were already known, then the messages back from the check node c_r to the variable nodes only consist of confirmations of the values (because we assumed that no errors took place — only erasures).

The purpose of having a Tanner graph is to illuminate this passing of messages.

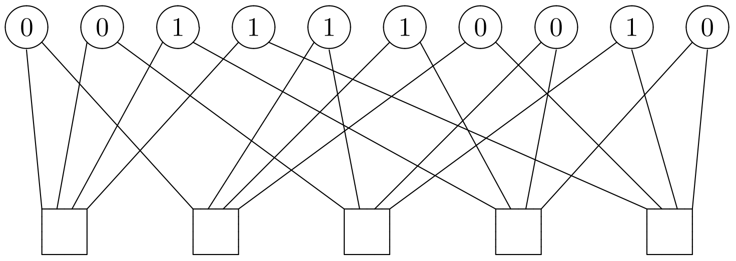
Example 5.3 *Assume that a word of the code of Example 5.1 is transmitted through a binary erasure channel, and is received in the form* ?011?10?1?. *We initialize the variable nodes in the Tanner graph of Figure 5.1 with these values.*



In the first half-iteration the check nodes receive respective messages (from left to right) ?011, ??10, 0??1, 11??, *and* 101?. *The three check nodes in the middle received two ?-messages each, and hence cannot solve any of the variables from their respective parity check equations. On the other hand the first check node only received a single* ?, *and can thus solve the value of* b_1 *from its knowledge of* b_2, b_3, b_4. *Thus it can send back one useful message: "your pals think that your value is* 0*" to variable node number* 1. *Similarly the last check node can solve* $b_{10} = 0$ *from its check equation. The first and the last bits contentedly update their contents. So after the first iteration the situation is thus as follows.*



The messages received by the check nodes during the first half of the second iteration are *thus* 0011*,* 0?10*,* 0??1*,* 11?0 *and* 1010*. The work of check nodes one and five is done. We observe that this time check node number two can solve* $b_5 = 1$*. Similarly check node number four can solve* $b_8 = 0$*, while check node number three can only twiddle its thumbs. The scene is now*



We see that the values of all the variable nodes are now known. However the algorithm may not know about it yet (depends how you code it!)? One option is to let the third check node finally give the decoded word its nod of approval. After all, the other check nodes are already content. This way we get an extra verification that we now have, indeed, a valid codeword.

A couple of general remarks about the general theory are due. Assume that we have a very long code. Assume that we further node the weight distributions of both the variable and the check nodes, in other words we know the fraction of variable nodes connected to one, two, et cetera check nodes, and that we similarly know the fraction of check nodes involving (one, two,) three, four, et cetera variables. The Tanner graph of our example is special in this sense that all the variable node are involved in exactly two parity check equations, and all the check equations involve exactly four bits. This type of Tanner graphs are called *regular*. Also assume that we know the probability of any of the bits getting erased. It is not unreasonable to think that under those circumstances we can predict how often the above algorithm will then stall (no progress during an iteration)/not stall, and thus fail/succeed. After that we can start playing the game of looking for optimal weight distributions of both variable and check nodes. While the analysis is somewhat simpler on a regular Tanner graph, it turns out that non-regular graphs often yield better codes. All of this is beyond the scope of this course.

5.2 Belief propagation, AWGN-channel, LDPC-codes

We next explain the necessary modifications to the message passing algorithm, when the messages involve a full continuum of reliability information. The first obstacle to a generalization is what can a check node do, when it is only fed reliability information. To that end we prove the following

Lemma 5.1 *Assume that the events* $(b_i = 0)$ *with* $i = 1, 2, \ldots, k$ *are independent and that*

$P(b_i = 0) = p_i$. Then the probability that $\sum_{i=1}^k b_i = 0$ is given by the formula

$$2P(\sum_{i=1}^{k} b_i = 0) - 1 = \prod_{i=1}^{k}(2p_i - 1).$$

Proof. We prove this by induction on the number of bits k, the base case $k = 1$ being obvious. So assume that for some natural number ℓ we have

$$2P(\sum_{i=1}^{\ell} b_i = 0) - 1 = f_\ell = \prod_{i=1}^{\ell}(2p_i - 1).$$

Then $s_\ell = \sum_{i=1}^{\ell} b_i$ vanishes with probability $(1+f_\ell)/2$ and $s_\ell = 1$ with probability $(1-f_\ell)/2$. The sum $\sum_{i=1}^{\ell+1} b_i$ vanishes when $s_\ell = b_{\ell+1}$. By the independence assumption this happens with probability

$$P(s_\ell = 0, b_{\ell+1} = 0) + P(s_\ell = 1, b_{\ell+1} = 1) = \left(\frac{1+f_\ell}{2}\right)p_{\ell+1} + \left(\frac{1-f_\ell}{2}\right)(1 - p_{\ell+1}),$$

so expanding the right hand side gives that

$$2P(s_\ell + b_{\ell+1} = 0) - 1 = f_\ell(2p_{\ell+1} - 1)$$

as required. ■

Here we make several observations. Let $\hat{b}$ be the so called hard decision value of a bit b, i.e. $\hat{b} = 0$, if $P(b = 0) \geq 1/2$ and $\hat{b} = 1$, if $P(b = 0) < 1/2$. Then the factor $2p_i - 1 \geq 0$ if and only if $\hat{b} = 0$. From the lemma we immediately see that $\sum_{i=1}^k \hat{b}_i$ is more probable as a value of $\sum_{i=1}^k b_i$, because the number $\prod_i(2p_i - 1)$ is positive if and only if an even number of the probabilities p_i are $< 1/2$. Furthermore, the factors $2p_i - 1 \in [-1, 1]$ and $|2p_i - 1| = 1$ only when there is a certainty about the value of b_i. Also we have

$$\left|\prod_{i=1}^{k}(2p_i - 1)\right| \leq |2p_j - 1|$$

for all indices $j = 1, 2, \ldots, k$. In other words we get a confirmation of the intuitively obvious fact that the uncertainty of the sum of uncertain bits is higher than the uncertainty of any of the bits participating in the sum.

We want to rewrite the result of Lemma 5.1 using log-likelihood ratios. If $LLR(b) = x$, then $P(b = 0) = e^x P(b = 1) = e^x(1 - P(b = 0) =$. From this equation we solve $P(b = 0) = e^x/(e^x + 1)$. Consequently

$$2P(b = 0) - 1 = 2\,\frac{e^x}{e^x + 1} - 1 = \frac{e^x - 1}{e^x + 1} = \tanh\frac{x}{2}.$$

Let us define a function

$$\phi(x) := \ln\left(\frac{e^x + 1}{e^x - 1}\right) = -\ln\tanh\frac{x}{2},$$

when x is a positive real number.

Lemma 5.2 *The function ϕ is an everywhere decreasing bijection from the set $\mathbf{R}_{>0}$ to itself. It is its own inverse, i.e.*

$$\phi(\phi(x)) = x$$

for all positive real numbers x.

Proof. A straightforward exercise. ∎

On some occasions it is convenient to extend the definition of ϕ by declaring $\phi(0) = \infty$ and $\phi(\infty) = 0$. This turns ϕ to a continuous function (with respect to the obvious topologies) from $[0, \infty]$ to itself.

Proposition 5.1 *Assume that the bits $b_i, i = 1, 2, \ldots, k$ are independent random variables, and that we know that $LLR(b_i) = x_i$. Then the most probable value of the sum $s = \sum_{i=1}^k$ is $\hat{s} = \sum_{i=1}^k \hat{b}_i$. The absolute value of its log-likelihood ratio can be calculated from the formula*

$$|LLR(s)| = \phi\left(\sum_{i=1}^k \phi(|x_i|)\right).$$

The sign of $LLR(s)$ is $(-1)^{\hat{s}}$.

Proof. Assume first that $\hat{b}_i = 0$ for all i. Then $x_i \geq 0$ for all i. By Lemma 5.1 we have

$$2P(s = 0) - 1 = \prod_{i=1}^k \frac{e^{x_i} - 1}{e^{x_i} + 1}.$$

Hence

$$\ln(2P(s = 0) - 1) = -\sum_{i=1}^k \phi(x_i).$$

Therefore

$$\phi(LLR(s)) = -\ln(2P(s = 0) - 1) = \sum_{i=1}^k \phi(x_i).$$

In this case the claim follows from the fact that ϕ is its own inverse.

If some of the hard decision values $\hat{b}_i$ are equal to one, i.e. $x_i < 0$, then we can invert those bits b_i. Such an inversion alters the parity of the sum s, replaces the factor $2P(b_i = 0) - 1$ with its negative, and similarly replaces x_i with its negative. Therefore a total of $\hat{s}$ sign changes take place, and the formula holds in the general case as well. ∎

Another problem we need to address is how to obtain the intrinsic LLRs for each bit. To say anything about this, we need to model the communication channel. A common and relatively realistic model is the so called additive white gaussian noise channel (AWGN for short). In the simplest case this means that a bit b is first mapped to a real number x as follows: $0 \mapsto +1, 1 \mapsto -1$. We then assume that the channel adds random noise n to the transmitted signal so that the receiver reads the number

$$y = x + n$$

from the channel. Here we assume that n is a normally distributed zero-mean random variable with known standard deviation σ. This means that the probability density function of n is

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{n^2}{2\sigma^2}}.$$

Therefore we get the conditional probabilities (in the sense of probability density)

$$P(y|b=0) = P(y|x=1) = P(n=y-1) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y-1)^2}{2\sigma^2}}$$

and similarly

$$P(y|b=1) = P(n=y+1) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y+1)^2}{2\sigma^2}}.$$

A priori we usually do not know anything about the value of b, i.e. $P(b=0) = P(b=1) = 1/2$. This is because either b is a message bit chosen by the user, or a check bit of a linear code (and thus again equally likely to have either value). There are some exceptional situations, e.g. when b is a terminating zero of a convolutional code, but in that case one may question the wisdom of spending transmission power in sending b anyway!

This allows us to reverse the order of the conditional probabilities as

$$P(b=0|y) = \frac{P(b=0,y)}{P(y)} = \frac{P(b=0)}{P(y)}\,\frac{P(y,b=0)}{P(b=0)} = \frac{1}{2P(y)}\,P(y|b=0)$$

and similarly

$$P(b=1|y) = \frac{1}{2P(y)}\,P(y|b=1).$$

Hence

$$LLR(b|y) = \ln\frac{P(b=0|y)}{P(b=1|y)} = \ln\left(\frac{e^{-\frac{(y-1)^2}{2\sigma^2}}}{e^{-\frac{(y+1)^2}{2\sigma^2}}}\right) = \frac{2y}{\sigma^2}.$$

Remember that the receiver can read y from the channel. In order to calculate the LLR, the receiver needs to also know the noise variance σ^2. The receiver can also measure this, because it has access to several thousands of transmitted bits, and can thus accurately estimate the expected value

$$E(y^2) = E((x+n)^2) = E(x^2) + 2E(xn) + E(n^2) = 1 + 0 + \sigma^2 = 1 + \sigma^2.$$

In what follows we thus assume that the receiver know the value of σ^2. As $E(x^2) = 1$ and $E(n^2) = \sigma^2$, and the energy of a radio signal (or any harmonic oscillation) is proportional to the square of the amplitude, the ratio $1/\sigma^2$ is often called the signal-to-noise ratio (=SNR).

We can finally describe the details of the message passing algorithm (also called the belief propagation algorithm) on a Tanner graph, when reliability information (i.e. LLRs) is used. The data structure needs to support the following items.

- Each variable node needs to know:
  - the intrinsic LLR of this bit,
  - the messages from all the check nodes this bit is involved in,
  - the best guess of the hard decision value of this bit (0 or 1).
- Each check node needs to know:
  - all the incoming messages (one from each bit participating in this check equation),

– (optionally) information about whether the hard decision values of the above bits pass this parity check.

At the beginning of the belief propagation algorithm the data at variable nodes is initialized in such a way that the values of the intrinsic LLRs are obtained from the receiver (based on the received signal according to the channel model and whatever else affects the interpretation of the signal). The message data is initialized to all zeros (no information has arrived from the check nodes yet).

During the first half of any iteration all the variable nodes send messages to all the check nodes connected to them in the Tanner graph. Assume that variable node i has intrinsic LLR x_i and messages $m(k,i)$ that it received from check node number k during the previous iteration. Then the message sent to check node number k will be the number $n(i,k) := x_i + \sum_{j,j\neq k} m(j,i)$. In other words, we take the information received from OTHER check nodes into account. The variable node also calculates the sum $z_i = x_i + \sum_j m(j,i)$ and updates the hard decision guess of bit b_i to be $\hat{b}_i = 0$, if $z_i \geq 0$, and $\hat{b}_i = 1$, if $z_i < 0$. Note that in practices it makes sense to first compute z_i, and the subtract the effect of a single message, so $n(i,k) = z_i - m(k,i)$. Optionally we can also send the value of $\hat{b}_i$ to all the check nodes.

During the second half of any iteration the check nodes process the incoming messages, and replies to them by sending each variable back information about the LLR of each bit that can be deduced from the LLRs of the other bits involved in this check equation. So check node number k sends the variable node number i the number

$$m(k,i) := \prod_{j\neq i} sgn(n(j,k))\ \phi\left(\sum_{j\neq i}\phi(|n(j,k)|)\right).$$

Optionally the check node may also send a central controller information about whether the hard decision values of the bits pass the parity check.

The algorithm terminates after a prescribed number of rounds (usually 50 or 100), or when all the check nodes report that the variable nodes' current beliefs about the bit values now pass all the parity checks, and further iterations are thus unnecessary.

Before describing the goal of all these calculations, let us take a look at a toy example.

Example 5.4 *Follow the belief propagation algorithm, when the repetition code of length three defined by the check matrix*

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

is used. We assume that the intrinsic LLR of bit number i is x_i, and initialize the Tanner graph accordingly.

During the first iteration the following messages are passed. The variable nodes send their intrinsic LLRs to the check nodes. Now that the check nodes are only connected to two variable nodes they reply by sending back to each variable node the LLR of the other bit involved in the parity check equation.

During the second iteration the messages sent by the first and the third variable nodes remain unchanged as they did not receive any messages independent from the lone check node they are connected to. On the other hand, the second variable node now takes into

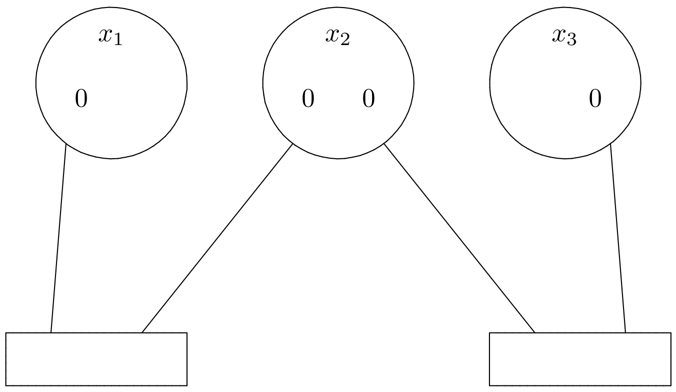


Figure 5.2: An initialization of belief propagation on a Tanner graph

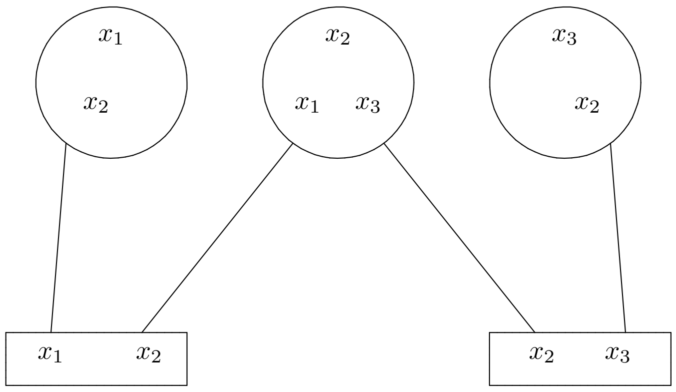


Figure 5.3: Belief propagation on a Tanner graph after a single iteration

account the message it received from the first check node in its message to the second check node, and vice versa. Therefore the message sent from the check nodes to the first and third variable nodes are updated accordingly, but no changes are made to the message sent to the second variable node. The situation after the second iteration is shown in Figure 5.4.

We observe that further iterations will no longer change the contents of the messages so we may as well stop after two iterations here.

Let us study the contents of the variable nodes at the end of the toy example run. We see that at this point for all $i = 1, 2, 3$ we have $z_i = x_1 + x_2 + x_3$. In other words at the end of the belief propagation all the bits think that they are equal to 0, if $x_1 + x_2 + x_3 \geq 0$ and all think that they are equal to 1, if $x_1 + x_2 + x_3 < 0$. In the repetition code all the bits agree, so this is certainly useful. Also all the bits base their final decision on the sum of their LLRs. What is the interpretation of this sum? Initially the variable nodes only had intrinsic information. The key point not taken into account was that the combination of the transmitted bits was a valid codeword with absolute certainty. As usual we assume that a

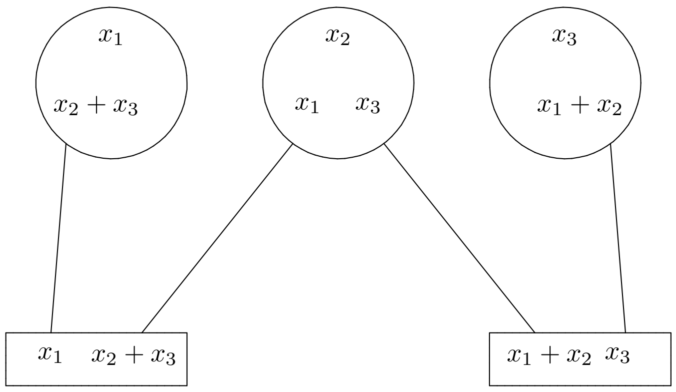


Figure 5.4: Belief propagation on a Tanner graph after two (or more) iterations

priori each codeword is transmitted with the same probability, and that the intrinsic LLRs are independent from each other. For example, if we transmit the three bits via an AWGN channel, separated in time, then we receive a vector $\mathbf{y} = (y_1, y_2, y_3)$, and $x_i = LLR(b_i|y_i)$. Independence means that

$$P(\mathbf{y}|000) = P(y_1|b_1 = 0) \cdot P(y_2|b_2 = 0) \cdot P(y_3|b_3 = 0),$$

and similarly

$$P(\mathbf{y}|111) = P(y_1|b_1 = 1) \cdot P(y_2|b_2 = 1) \cdot P(y_3|b_3 = 1).$$

Here in the conditional probabilities the events 000 and 111 should be read as "000 (resp. 111) was transmitted", and the event $\mathbf{y}$ should be read as "the vector $\mathbf{y}$ was received". The assumption that the codewords were a priori equally likely to be transmitted implies that

$$\ln\left(\frac{P(000|\mathbf{y})}{P(111|\mathbf{y})}\right) = \ln\frac{P(b_1 = 0|y_1)}{P(b_1 = 1|y_1)} \ln\frac{P(b_2 = 0|y_2)}{P(b_2 = 1|y_2)} \ln\frac{P(b_3 = 0|y_3)}{P(b_3 = 1|y_3)} = x_1 + x_2 + x_3.$$

Because all the bits are known to be equal in our case we can interpret this as saying that

$$\ln\left(\frac{P(b_i = 0|\mathbf{y})}{P(b_i = 1|\mathbf{y})}\right) = x_1 + x_2 + x_3.$$

In other words, taking the entire received vector into account, we gain the information from the repeated bits. This motivates the following definition.

Definition 5.1 *Assume that a binary code $\mathcal{C}$ is being used. Denote its codewords by* $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. *Given the received vector* $\mathbf{y}$ *(and a channel model) we call the LLR*

$$LLR(b_i = 0|\mathbf{y}) = \ln\left(\frac{\sum_{\mathbf{b}\in C, b_i=0} P(\mathbf{y}|\mathbf{b})}{\sum_{\mathbf{b}\in C, b_i=1} P(\mathbf{y}|\mathbf{b})}\right)$$

the a posteriori *LLR of bit b_i.*

The a posteriori probabilities for the values of the bit b_i can be interpreted as follows. We split the code $\mathcal{C}$ into two piles of codewords according to the value of the b_i. Then we calculate the log-likelihood ratio of the two piles conditioned on the received vector. Of course, when $\mathcal{C}$ is large, calculating this LLR by brute force means appears to have prohibitive complexity. However, we have the following result.

Theorem 5.1 *Assume that the Tanner graph of a code is a tree, i.e. it contains no cycles. Then at the end of the belief propagation algorithm we have that each variable node can calculate the a posteriori LLR of its bit as the sum of the intrinsic LLR and the incoming messages from the check nodes:*

$$z_i = x_i + \sum_k m(k,i) = LLR(b_i = 0|\mathbf{y}).$$

Proof. Omitted. ∎

The key assumption in the previous theorem is that the Tanner graph should have no cycles. Let us alter our toy example a bit to see the effect of having a cycle in the Tanner graph.

Example 5.5 *Let us add a third, redundant check equation to the definition of the repetition code. So the parity check matrix looks like*

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

and the corresponding initialized Tanner graph is shown in Figure 5.5

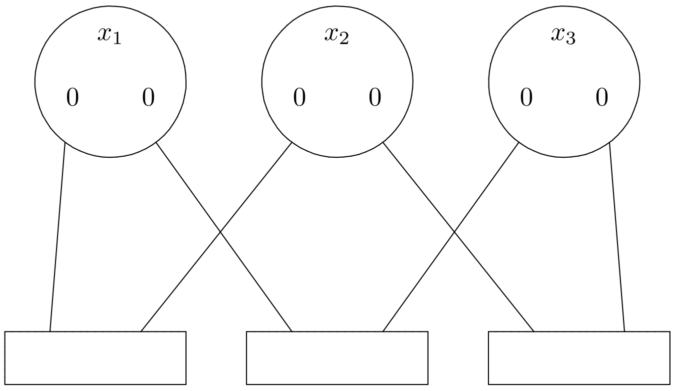


Figure 5.5: Belief propagation on a Tanner graph with a 6-cycle

As earlier, we see that after a single iteration, the check nodes have sent messages (x_2, x_3) to the first variable node, messages (x_1, x_3) to the second, and (x_1, x_2) to the third. The second iteration updates the messages from the check nodes to (x_2+x_3, x_2+x_3) to the first variable, (x_1+x_3, x_1+x_3) to the second and (x_1+x_2, x_1+x_2) to the third. The third iteration sees only messages equal to $x_1 + x_2 + x_3$ going from all the check nodes to their neighboring variable *nodes. In the fourth iteration the first variable node receives* $(x_1 + 2x_2 + x_3, x_1 + x_2 + 2x_3)$, *the second receives* $(2x_1 + x_2 + x_3, x_1 + x_2 + 2x_3)$ *and the third* $(2x_1 + x_2 + x_3, x_1 + 2x_2 + x_3)$. *At this point the LLR-totals at all the variable nodes are equal to* $3x_1 + 3x_2 + 3x_3$. *We see that as iterations tick by the coefficients will keep increasing.*

We interpret the outcome of the last toy example as saying that the presence of cycles in the Tanner graph creates dependencies among the messages - the same information is fed back to the messages after they have completed a full cycle, reinforcing themselves in an unintended manner. However, we also observe that this phenomenon does not lead to the wrong decisions made at the end. The reinforcing cycles scale the a posteriori LLRs incorrectly, but (often enough) do not change the signs. Therefore we can use the belief propagation algorithm even if the Tanner graph contains some cycles. A more careful analysis shows that the shorter the cycle the more detrimental its effect may be. Therefore code designers take great pains to exclude all cycles of length 4, and usually as many, if not all, of the cycles of length 6 as they can.

A few words about LDPC-codes. Contrary to our toy examples the more realistic parity check matrices have a very low number of 1s, indeed. As an example I mention an LDPC code of length $n = 64800$ that will be used in the DVB-T2 standard (the update to the current digital TV system used in Europe including Finland). The code has dimension 43200, so there will be 64800 variable nodes and 21600 check nodes in the Tanner graph. Of the variable nodes 4320 are connected to 13 check nodes, 38880 to 3 check nodes, 21599 to two check nodes, and there is a single variable node connected to only one check node. The average number of check nodes a variable node is connected is thus 10/3. So the check matrix H has in the average a tad more than 3 ones on a column with 21600 entries! The average number of bits involved in a parity check is thus ten. It stands to reason that these number should be low. If there are many variable nodes connected to a check node, then that check node will usually only send relatively weak messages given that out of several participating bits, some will be unreliable, and as we saw, the unreliability adds up. Consequently the number of connections per variable node has to be even lower, because we cannot have too many independent parity checks.

Optimal distributions of the weight distributions on the Tanner graph can be found using tools from stochastics. We cannot study that theory within the scope of this course. Nevertheless, very good LDPC-codes (of a prescribed rate) can be designed with the known tools. A catch is that for the codes to work as well as promised by the theory (that is asymptotic in nature) they need to be relatively long.

5.3 Turbo codes, Soft output Viterbi algorithm

Originally (some variations are possible) turbo codes are binary linear codes of rate 1/3 gotten from a $(2, 1)$ convolutional code $\mathcal{C}$ by the following recipe. First we find a systematic generator matrix $G_{sys}(D) = (1, f(D))$ for C. This forces a rational function as the latter component $f(D)$. For example with our favorite convolutional code we use the generator matrix $G_{sys}(D) = (1, (1 + D^2)/(1 + D + D^2))$ instead of the polynomial generator matrix $G(D) = (1 + D^2, 1 + D + D^2)$. To turn $\mathcal{C}$ into a turbo code we need to specify the length of the input sequence $N = \deg u(D) + 1$. In addition to that we need a so called *interleaver*, i.e. a bijection $\sigma \in Sym(\{0, 1, 2, \ldots, N - 1\})$ permuting the indices $i = 0, 1, 2, \ldots, N - 1$.

The key idea in turbo codes is to use the same convolutional code $\mathcal{C}$ also to encode the interleaved message $\tilde{u}(D) = \sum_{i=0}^{N-1} u(\sigma(i))D^i$. The words of the turbo code are the vectors $(u(D), u(D)f(D), \tilde{u}(D)f(D)) \in F[[D]]^3$. A couple of remarks are due. Observe that we do not transmit the sequence $\tilde{u}(D)$. This would be kind of wasteful, because it would amount to repeating the bits of $u(D)$ albeit in a different order. As $f(D)$ is no longer a polynomial function, terminating the power series $u(D)f(D)$ and $\tilde{u}(D)f(D)$ (i.e. turning them into polynomials) usually requires non-zero dummy input bits. Furthermore, the two power series usually require different sequences of dummy bits for termination. There are some variations on how to do this, but as long as the decoder is aware of the exact method, the choice doesn't really matter.

The performance of a turbo code depends on the following ingredients. A good choice of the interleaver π allows (as we shall see) us to get a code with a higher Hamming distance. On the other hand the presence of convolutional codes allows us to construct an iterative efficient (soft decision) decoding algorithm. We cannot quite use the ordinary Viterbi algorithm, because its output consists of hard information - the closest codeword. Luckily a more suitable version is available. To decode the turbo code iteratively we need a variant of the Viterbi algorithm that calculates the a posteriori LLRs of individual component bits of $\mathbf{u}(D)$. Again we utilize the trellis. Namely, in the formula for the a posteriori probabilities it is natural to lump together the codewords according to the state of the trellis that they pass through just before and immediately after the bit of interest. This is the idea behind the so called two-way algorithm (also known as the forward-backward algorithm).

Let us concentrate on a single trellis section between instances of time i and $i+1$. I denote by $\Sigma(i)$ the state space at instant i. Assume that we have already calculated, conditioned on all a priori information about the bits $u(j), j < i$ and the intrinsic information about all the bits $x_\ell(j), j < i, \ell = 1, 2$, that the probabilities of the early part of the path of a codeword ends in the state $\mathbf{s} \in \Sigma(i)$ is $P^-(\mathbf{s}, i)$. Assume that we have similar information about the probability $P^+(\mathbf{s}, i)$, based on the intrinsic information about the bits $x_\ell(j), j \geq i+1, \ell = 1, 2$, and on a priori information about the input bits $u(j), j \geq i+1$, of a valid codeword continuing from the state $\mathbf{s} \in \Sigma(i+1)$ in the future.

For each edge e in this trellis section we assign a number $P(e) = P(in(e))P(out(e))$ that is the a priori probability $P(in(e))$ of its input label, $in(e)$, times the probability of its output label $P(out(e))$. As we assume that the random processes involving the reception of all the bits are independent from each other, this is just the product of the probabilities of the individual bits of $out(e)$ conditioned on the received version of this part of the codeword. Then we can calculate the following probabilities:

- The probabilities $P^-(\mathbf{s}, i+1)$ for all the states $\mathbf{s} \in \Sigma(i+1)$.
- The probabilities $P^+(\mathbf{s}, i)$ for all the states $\mathbf{s} \in \Sigma(i)$.
- The ratio of the a posteriori probabilities $P(u(i) = 0)$ and $P(u(i) = 1)$ conditioned on the available data.

Before we look at the formulas for all these probabilities, I remark that the probabilities $p^-(\mathbf{s}, i)$ are calculated *forward*, i.e. in the direction of increasing i. The probabilities $P^+(\mathbf{s}, i)$ on the other hand are calculated *backward*, in the direction of decreasing i. We shall see that the formula for the a posteriori probabilities of the input bits requires all P^+ and P^-

to be known. So we need to traverse the trellis twice (hence the name forward-backward algorithm), the a posteriori probabilities can be calculated on the latter direction.

On with the formulas. Any state $\mathbf{s} \in \Sigma(i+1)$ can be reached from a state $\mathbf{s}' \in \Sigma(i)$, iff there are (one or more) edges going from $(\mathbf{s}', i)$ to $(\mathbf{s}, i+1)$. We denote the set of such edges by $E(\mathbf{s}', \mathbf{s}, i)$. Note that the set is usually empty for some states $\mathbf{s}'$. As in the usual Viterbi algorithm we add the effect of an edge e to what we know about its starting initial state $\in \Sigma(i)$ to get something about its final state $\in \Sigma(i+1)$. As this time both pieces are probabilities of independent events, we *multiply* the two numbers instead of adding them (as was done in vanilla Viterbi). Also, because we want to take all the paths through the trellis into account, we *add* the atomic probabilities together instead of selecting the path with minimum total distance (as was the case in vanilla Viterbi). So in the end we get, for each state $\mathbf{s} \in \Sigma(i+1)$ the following probability mass

$$M^-(\mathbf{s}, i+1) = \sum_{\mathbf{s}' \in \Sigma(i)} \sum_{e \in E(\mathbf{s}', \mathbf{s}, i)} P^-(\mathbf{s}', i) P(e).$$

There is no reason for these sums to sum up to one. So we can (but actually don't necessarily have to) rescale them to meet that criterion (this is one way of conditioning these probabilities to the fact we only take into account paths corresponding to valid codewords). So let $M^-(i+1) = \sum_{\mathbf{s} \in \Sigma(i+1)} M^-(\mathbf{s}, i+1)$. Then the sought probabilities are

$$P^-(\mathbf{s}, i+1) = \frac{M^-(\mathbf{s}, i+1)}{M^-(i+1)}.$$

The backward calculation is completely analogous (imagine that you reverse the direction of time, and walk through the trellis in the opposite direction). So we calculate the probability masses

$$M^+(\mathbf{s}', i) = \sum_{\mathbf{s} \in \Sigma(i+1)} \sum_{e \in E(\mathbf{s}', \mathbf{s}, i)} P(e) P^+(\mathbf{s}, i+1),$$

their sum $M^+(i) = \sum_{\mathbf{s}' \in \Sigma(i)} M^+(\mathbf{s}', i)$, and (using that as a normalization factor) the probabilities

$$P^+(\mathbf{s}', i) = \frac{M^+(\mathbf{s}', i)}{M^+(i)}.$$

In order to calculate the a posteriori likelihood ratio (or LLR) of the input bit $u(i)$ we, as usual, split codewords into two groups according to the value of $u(i)$. We further split them into smaller groups according to the states the codewords pass through at instants i and $i+1$. As the structure of the code allows us to join any ending from $i+1$ to ∞ with any beginning from $-\infty$ to i, we are all set, and the definition of the a posteriori probability gives us the formula

$$LLR(u(i)) = \ln \frac{\sum_{\mathbf{s}' \in \Sigma(i)} \sum_{\mathbf{s} \in \Sigma(i+1)} \sum_{e \in E(\mathbf{s}', \mathbf{s}, i), in(e)=0} P^-(\mathbf{s}', i) P(e) P^+(\mathbf{s}, i+1)}{\sum_{\mathbf{s}' \in \Sigma(i)} \sum_{\mathbf{s} \in \Sigma(i+1)} \sum_{e \in E(\mathbf{s}', \mathbf{s}, i), in(e)=1} P^-(\mathbf{s}', i) P(e) P^+(\mathbf{s}, i+1)}.$$

Observe that if we forget to scale the probability masses $M^-(\mathbf{s}', i)$ and $M^+(\mathbf{s}, i+1)$ to sum up to unit, and simply use them in place of $M^-(\mathbf{s}', i)$ and $M^+(\mathbf{s}, i+1)$, then all the terms in both numerator and the denominator will be multiplied by $M^-(i)M^+(i+1)$, and this factor is thus cancelled in the ratio. Therefore the scaling is not always done. However, in a long trellis the numbers then quickly become quite small. Depending on the hardware carrying out the calculations this may or may not be a problem.

Example 5.6 *Solutions to the homework problems 6 and 7 from the last set of 2013 version of the course will be added here.*

Now we can describe an iterative decoding algorithm for a turbo code. We initialize the a priori LLRs for the input bits $u(i)$ to be all zero. We can calculate the probabilities of all the edges $P(out(e))$ for both component convolutional codes: the first code that transmitted $(u(D), u(D)f(D))$ and the second code that transmitted $(\tilde{u}(D), \tilde{u}(D)f(D))$ - conditioned on the received signal only. Observe that the intrinsic information on the bits of $\tilde{u}(D)$ is the same as that of the bits $u(D)$.

Then we can iterate the following two stage algorithm a fixed number of times.

(i) Run SOVA on the first component code. Update the a priori LLRs of $u(i)$ to be equal to the freshly calculated a posteriori LLRs.

(ii) Run SOVA on the second component code (observe that we now use the a posteriori LLRs of the input bits of the first component code as a priori LLRs). Update the a priori LLRs of $u(i)$ to be equal to the freshly calculated a posteriori LLRs.

Observe that the two SOVA-decoders take turns using the a posteriori LLRs provided by the other as a priori LLRs. The two steps here are called half-iterations.

The idea is that the two decoders keep feeding each other with progressively more accurate information about the transmitted word. We can loosely interpret this in the same spirit as the belief propagation algorithm as follows. The first half iteration attempts to decode the bit $u(i)$ using its nearby neighbors $u(i \pm 1), u(i \pm 2), \ldots, u(i \pm m)$. This is because in the trellis we can reach any state from a given one in m steps, and at approximately that point an eventual disturbance caused by an incorrect bit has been corrected (it would take too long to justify this). Similarly in the second half iteration we take into account information from bits $u(\sigma^{-1}(\sigma(i) \pm k)), k = 1, 2, \ldots, m$, to get more accurate information about the bit $u(i) = u(\sigma^{-1}(\sigma(i)))$. If the interleaver σ is designed well enough the two sets of bits do not intersect for any i, and thus the two half-iterations give independent information. As we carry out more iterations we gradually collect information about $u(i)$ from further and further out (much like in the case of LDPC codes).

Another thing we must take into account when designing an interleaver is the minimum Hamming distance of the resulting turbo code. A carelessly chosen interleaver may create a turbo code with several low weight words. These will hurt the performance of a turbo code by creating a so called error floor. This is a well studied problem, but we end our cursory description of turbo codes here.