Appendix B - Finite Fields and Coset Codes

Appendix B

Finite Fields and Coset Codes

Sections B.3-B.6 may look a little out of place here and correspond to an older version where these trellis and coset codes were in the main coding chapter. They are presented here for completeness, but largely superseded by LDPC, turbo, and polar codes.

B.1 Finite Field Algebra

Coding theory uses the concepts of finite fields, algebras, groups and rings. This brief appendix concisely reviews the basics of these topics.

B.1.1 Groups, Rings, and Fields

A group is a set of objects, that is closed under an operation addition, associative over that same operation, and for which an identity and inverse exist in the group. More formally,

Definition B.1.1 *[Group] A* **group** *S is a set, with a well-defined operation for any two members of that set, call it* **addition** *and denote it by $+$, that satisfies the following four properties:*

1. **Closure** $\forall\ s_1, s_2 \in S$, *the sum* $s_1 + s_2 \in S$.
2. **Associative** $\forall\ s_1, s_2, s_3 \in S$, $s_1 + (s_2 + s_3) = (s_1 + s_2) + s_3$.
3. **Identity** *There exists an identity element* 0 *such that* $s + 0 = 0 + s = s$, $\forall\ s \in S$.
4. **Inverse** $\forall\ s \in S$, *there exists an inverse element* $(-s) \in S$ *such that* $s + (-s) = (-s) + s = 0$.

The identity element 0 is unique. When the group also exhibits the **commutative property**, $s_1 + s_2 = s_2 + s_1$, the group is said to be an **Abelian** group. A **subgroup** is a subset of S that satisfies all the properties of a group.

A ring is an Abelian group with the additional operation of multiplication, such that closure and associativity also hold for multiplication, and that multiplication distributes over addition. More formally,

Definition B.1.2 (Ring) *A* **ring** *R is an Abelian group, with the additional well-defined operation for any two members of that set, call it* **multiplication** *and denote it by $\cdot$ (or by no operation symbol at all), that satisfies the following three properties:*

1. **Closure for multiplication** $\forall\ r_1, r_2 \in R$, *the product* $r_1 \cdot r_2 \in R$.
2. **Associative for multiplication** $\forall\ r_1, r_2, r_3 \in R$, $r_1 \cdot (r_2 \cdot r_3) = (r_1 \cdot r_2) \cdot r_3$.
3. **Distributive** $\forall\ r_1, r_2, r_3 \in R$, *we have* $r_1 \cdot (r_2 + r_3) = r_1 \cdot r_2 + r_1 \cdot r_3$ *and* $(r_1 + r_2) \cdot r_3 = r_1 \cdot r_3 + r_2 \cdot r_3$.

A ring often has a **multiplicative identity** denoted by 1, and if multiplication is commutative, the ring is called a **commutative ring**. Any element of a ring R, call it r, for which a multiplicative inverse $1/r$ also exists in R is called a **unit** or **prime**. A field is a ring that defines division:

Definition B.1.3 *[Field] A* **field** *F is a ring, with the additional operation of division, the inverse operation to multiplication, denoted by $/$. That is for any $f_1, f_2 \in F$, with $f_2 \neq 0$, then $f_1/f_2 = f_3 \in F$, and $f_3 \cdot f_2 = f_1$.*

A somewhat weaker version of division occurs in what is known as the **integral domain**, which is a ring with the following additional property: $f_1 \cdot f_2 = f_1 \cdot f_3$ implies $f_2 = f_3$ if $f_1 \neq 0$.

A field may contain a finite or infinite number of member objects. A field of interest in this chapter is the finite field with two elements $GF(2) = \{0, 1\}$, with addition defined by $0 + 0 = 0$, $0 + 1 = 1$, and

$1+1=0$, multiplication defined by $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, and $1 \cdot 1 = 1$. The only unit or prime in $GF(2)$ is 1.

Another example of a field is $F(D)$ defined in Section 8.1, the ratios of all binary polynomials in D, where multiplication and division are defined in the obvious way, with modulo 2 addition.

Vector spaces are used often in this text in other Chapters and there often refer to vectors of real or complex numbers. More generally, and specifically, in coding, the vector space can have elements in any field, in particular a fine field.

Definition B.1.4 *[Vector Space] An n-dimensional* **Vector Space** *V over a field F contains elements called vectors $\boldsymbol{v} = [v_{n-1},\ ...,v_0]$, each of whose components v_i $i = 0,...,n-1$ is itself an element in the field F. The vector space is closed under addition (because the field is) and also under scalar multiplication where $f_i \cdot \boldsymbol{v} \in V$ for any element $f_i \in F$ where*

$$f_i \boldsymbol{v} = [f_i \cdot v_{n-1}, ..., f_i \cdot v_0] \ \ . \tag{B.1}$$

The vector space captures the commutativity, associativity, zero element (vector of all zero components), and additive inverse of addition and multiplication (by scalar of each element) of the field F. Similarly, the mulitplicative identity is the scalar $f_i = 1$. A set of J vectors is linearly independent if

$$\sum_{j=1}^{J} f_j \cdot \boldsymbol{v}_j = 0 \tag{B.2}$$

necessarily implies that

$$f_j = 0 \ \ \forall j \quad . \tag{B.3}$$

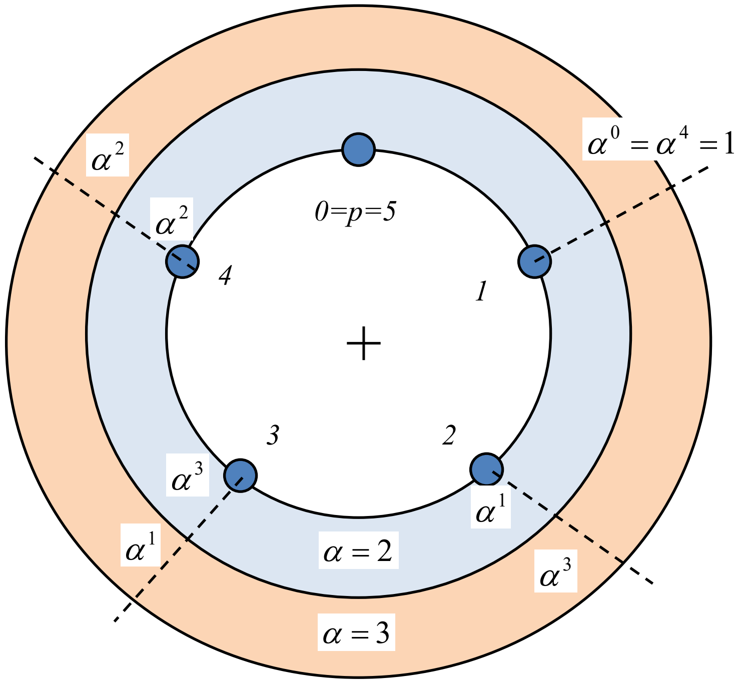
B.1.2 Galois Fields

Galois Fields are essentially based on arithmetic modulo a prime number p (or a power of a prime number p^m as to be shown shortly). The elements of a Galois field can be written

$$GF(p) = \{0, 1, ..., p-1\} \quad . \tag{B.4}$$

The notation $\mathbb{F}_p \stackrel{\Delta}{=} GF(p)$ often also finds use in this text and elsewhere. The simplest Galois Field $GF(2)$ uses binary arithmetic, where the prime is $p = 2$ and the elements are 0 and 1. Addition and subtraction reduce to binary "exclusive or," multiplication is binary "and," while division for non-zero elements is trivially 1/1=1. Figure **??** illustrates a less trivial example for $p = 5$, or $GF(5)$. In Figure **??**, a modulo-arithmetic circle illustrates addition, consistent with this text's use of modulo addition. Addition corresponds to moving clockwise around the circle while subtraction is counter clockwise. A multiplication table also appears in Figure **??**. This choice of multiplication definition simply multiplies integers and then takes the result modulo 5. Each row or column of this symmetric multiplication table contains each element of $GF(5)$ only once, which means that the reciprocal of an element is the column index for the entry 1 in the corresponding row of that element. (Division then occurs by multiplying by the reciprocal.) This reciprocal is the unique multiplicative inverse needed for a field. This subsection in particular consolidates many concepts introduced by Prof John Gill in his Stanford error-correcting-codes class [1].

$GF(5) = \{0\ \ 1\ \ \alpha\ \ \alpha^2\ \ \alpha^3\ \}$



×	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

α	α^2	α^3	α^4
2	4	3	1
3	4	2	1

Figure B.1: Illustration of $GF(5)$ addition- and multiplication-groups' closures.

Lemma B.1.1 *[GF(p)] The elements* $0,\ 1,\ \ldots\ p-1$ *of* $GF(p)$ *form a field under addition and multiplication modulo* p*, where* p *is prime.*

Proof: Closure of addition, subtraction (additive inverse), closure of multiplication, and the zero and identity element (1) follow trivially from the properties of integers, as do commutative and associative properties (all modulo p). Division and the multiplicative inverse do not trivially follow. A multiplication table has each row and column containing all the non-zero elements exactly once: This completeness of a row (or column) follows from observing that multiplication of an element $0 < \alpha \leq p-1$ by two distinct elements $0 < a_1 \leq p-1$ and $0 < a_2 < a_1$ cannot create the same result modulo p, for if they did

$$\alpha \cdot a_1 = p \cdot d_1 + r \tag{B.5}$$

$$\alpha \cdot a_2 = p \cdot d_2 + r \quad . \tag{B.6}$$

Equivalently,

$$\alpha \cdot (a_1 - a_2) = p \cdot (d_1 - d_2) > 0 \text{ and } = (0)_p \quad . \tag{B.7}$$

For such an equality to hold true, noting that $\alpha < p$ and also $0 < a_1 - a_2 < p$ since a_1 and a_2 are distinct, then neither the first or second term on the left can equal p and both are less than p. This necessarily implies $0 < d_1 - d_2 \leq \min(\alpha, (a_1 - a_2))$ and thus

$$\frac{\alpha(a_1 - a_2)}{d_1 - d_2} = p \tag{B.8}$$

a factorization of a prime number and a contradiction. Thus, each element can occur only once in each row and column if p is prime. **QED.**

Galois Fields often equivalently represent their nonzero elements by power of a primitive element α

$$GF(p) = \{0, 1, \alpha^1, \alpha^2, ..., \alpha^{p-2}\} \quad . \tag{B.9}$$

Figure B.1 illustrates the use of both $\alpha = 2$ and $\alpha = 3$ to generate $GF(5)$. The successive powers of each such **primitive element** (primative elements are prime numbers in this case) visits each and every non-zero $GF(5)$ element exactly once before $\alpha^{p-1} = 1$, and $\alpha^4 = 1$ in both cases. The element $\alpha = 4$ does not generate the entire field and instead generates $\{0, 1, \alpha = 4\}$, a subfield of 3 elements (which is $GF(3)$ where addition is refined with these symbols as $1 + 1 = \alpha$, $\alpha + \alpha = 1$, and $1 + \alpha = 0$ essentially redefining the symbol 4 to be 2. This "symbol interpretation" of the Galois Field is often good to keep in mind as addition and multiplication need not necessarily be defined in a direct correspondence with integer addition and multiplication (even though this example has done so outside of this observation on $GF(3)$ as a subfield of $GF(5)$ for the elements 0, 1, and 4. The multiplication table in Figure B.1, and thus multiplication, is invariant to the use of the prime α as 2 or 3.

More generally for $GF(p)$, the multiplicative identity is $\alpha^0 = 1$ for all elements $\alpha \in GF(p)$. The $(p-1)^{th}$ power of any element must always be unity, $\alpha^{p-1} = 1$ since the field contains p elements and thus a non-zero element's powers must repeat some nonzero value once a maximum $p - 1$ non-zero elements have been generated. This value that first repeats must be 1 by the uniqueness of the inverse already established. From the uniqueness of the rows and columns of the multiplication table, any prime-integer $\alpha > 1$ in $GF(p)$ is a primitive element and can be selected for the value of α to generate all the other nonzero $GF(p)$ elements $\alpha^0 = 1$, α^1, α^2, ..., α^{p-2}. A non-prime integer has the liability of being the product of primes, so that each movement corresponding to another multiplication by α in going from α^i to α^{i+1} actually corresponds to multiplication by each element in this non-prime's factorization (and so the repeating of the sequence occurs earlier because there are more steps implicit in this multiplication (or in fact more than one multiplication) and so "we get to 1 faster."

The use of the notation α^i is useful for multiplication because the exponents can be added, so

$$\alpha^i \cdot \alpha^j = \alpha^{(i+j)_{(p-1)}} \quad . \tag{B.10}$$

Adding of exponents is executed mod $p - 1$ because $\alpha^{p-1} = 1 \; \forall \alpha \in GF(p)$. Further, division is executed by multiplying by the inverse $\alpha^{-i} = \alpha^{p-1-i}$ so simple addition on exponents allows all multiplication and division within the field. Storage of each element's index as a power of α and its inverse's index as a power of α, along with simple mod-p addition or subtraction allows all computation with minimal computational effort. If addition is thus viewed as trivial, then perhaps the circle in Figure B.1 is more useful. Each multiplication by α consistently refers to rotation by 90 degrees clockwise in the figure (and division by α is rotation by 90 degrees in the opposite direction).

The use of Galois Fields is perhaps most useful when extended to vectors of up to $2^m - 1$ elements (the elements are typically vectors of m bits/pits). Such a Galois Field is denoted $GF(p^m)$ where p is prime. This field has p^m distinct elements. The elements of $GF(p^m)$ are m-dimensional vectors or m-tuples of $GF(p)$ elements. Typically $p = 2$ so these become vectors of bits, typically with $m = 8$ (so bytes in a digital system). However, any prime value of p and any positive integer m defines a Galois Field. It is convenient in such fields to think of an element as represented by a polynomial

$$u(D) = u_0 + u_1 \cdot D + u_2 \cdot D^2 + ... + u_{m-1}D^{m-1} \tag{B.11}$$

where $u_i \in GF(p) \; \forall \; i = 0, ..., m$. The powers of the variable D are used to represent positions within the vector, and multiplication by D corresponds to a shift of the elements (dealing with D^m will be addressed shortly). Addition and multiplication in $GF(p^m)$ is modulo a degree-m prime polynomial $g(D)$ with similarly $\alpha_i \in GF(p)$, and necessarily $g_0 \neq 0$ and $g_{m-1} \neq 0$. Thus, a remainder $r(D)$ modulo $g(D)$ with $deg(r(D)) < m$ represents any $u(D)$ with $deg(u(D)) \geq m$, or with $q(D)$ also a degree m or less polynomial with coefficients in $GF(p)$:

$$u(D) = q(D) \cdot g(D) \cdot +r(D) \quad . \tag{B.12}$$

Equivalently

$$\left(u(D)\right)_{g(D)} = r(D) \quad . \tag{B.13}$$

Multiplication and addition are based on $GF(p)$ multiplication and additon, respectively, of polynomial coefficients in $GF(p)$ to create new coefficients in $GF(p)$ but then followed by a modulo-$g(D)$, which follows by simple substitution of $g(D) = 0$ in the expression, so for instance if

$$u(D) \cdot v(D) = f(D) = \sum_{i=0}^{deg(f)} f_i \cdot D^i \tag{B.14}$$

then powers of $D^{i \geq m-1}$ are found by (potentially multiple uses of) the substitution $D^{m-1} = 1 + g_1 \cdot D + ... + g_{m-2} \cdot D^{m-2}$ that causes $g(D) = 0$. This concept of multiplication differs significantly from typical binary multiplication in digital computers where the multiplication of two 8-bit quantities would produce a 16-bit result. In Galois fields, multiplication of two 8-bit quantities produces another 8-bit quantity.

EXAMPLE B.1.1 *[$GF(2^2) = \mathbb{F}_4$ and $GF(2^3) = \mathbb{F}_8$]* The prime binary polynomial of degree 2 is $g(D) = 1 + D + D^2$ with degree-1 elements $\{0, 1, D, 1 + D\}$. The addition table is:

⊕	0	1	D	1+D
0	0	1	D	1+D
1	1	0	1+D	D
D	D	1+D	0	1
1+D	1+D	D	1	0

⊕	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

The addition table's entries The field elements D and $1 + D$ are interchangeable. Any $\mathbb{F}_4$ element is its additive inverse. The reader may observe this generally true for any $\mathbb{F}_{2^m}$ because the field elements are themselves binary polynomials. For this reason in $\mathbb{F}_{2^m}$, as with binary subtraction, the addition and subtraction operations are interchangeable. Referral to Figure B.1 shows that equivalence of addition and subtraction does not necessarily hold for Galois Fields based on $p \neq 2^m$.

The primitive elements of $\mathbb{F}_4$ are $\alpha^1 = D$ and $\alpha^2 = D^2 = 1 + D$ (because the operation modulo $g(D)$ will set $D^2 = 1 + D$). $\mathbb{F}_4$ multiplication tables are

⊗	0	1	D	1+D
0	0	0	0	0
1	0	1	D	1+D
D	0	D	1+D	1
1+D	0	1+D	1	D

or (lsb first)

⊗	00	10	01	11
00	00	00	00	00
10	00	10	01	11
01	00	01	11	10
11	00	11	10	01

or (lsb last)

⊗	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

The writing of binary polynomials in ascending power of D is consistent with Matlab's lowest entry (lsb) on left (first) instead of right (last). Writing the polynomial in descending power of D is consistent with this text's highest index at top/left convention. The polynomial approach actually provides flexibility in translation to either. Figure B.2 illustrates only $\mathbb{F}_4$'s multiplicative group with prime elements around the boundary.

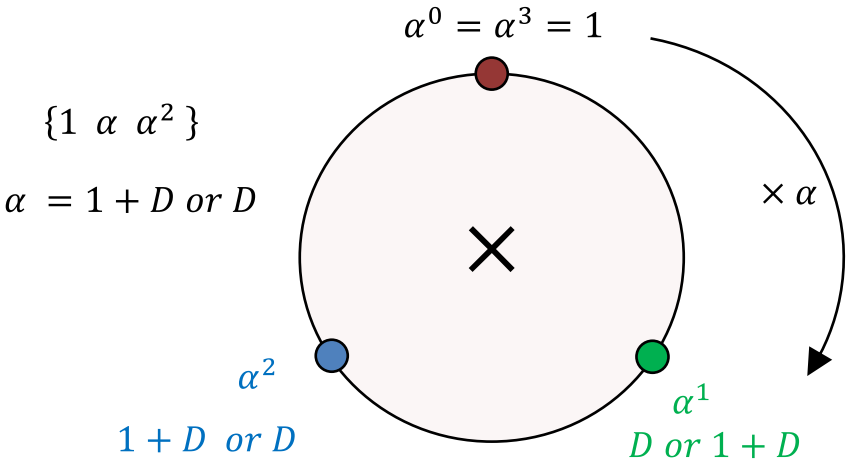


Figure B.2: Multiplicative generation of nonzero elements in $GF(5)$.

$\mathbb{F}_8$ repeats a similar exercise. Since $1 + D^7 = (1 + D) \cdot (1 + D + D^3) \cdot (1 + D^2 + D^3)$, there are two choices for the prime polynomial. This example choses $g(D) = 1 + D + D^3$. The primitive elements of degree 2 or less are D, $1 + D$, and $1 + D + D^2$, any one of which can be used to definite multiplication, so this example selects $\alpha^1 = D$. $\mathbb{F}_8$'s elements are therefore:

i	GF(8) element		
	α^i	lsb first	lsb last
$-\infty$	0	000	0
0	1	100	1
1	D	010	2
2	D^2	001	4
3	$1 + D$	011	6
4	$D + D^2$	110	3
5	$1 + D + D^2$	111	7
6	$1 + D^2$	010	5

The corresponding addition table with lsb FIRST is:

$\oplus$	0	1	D	D^2	$1 + D$	$D + D^2$	$1 + D + D^2$	$1 + D^2$
0	0	1	D	D^2	$1 + D$	$D + D^2$	$1 + D + D^2$	$1 + D^2$
1	1	0	$1 + D$	$1 + D^2$	D	$1 + D + D^2$	$D + D^2$	D^2
D	D	$1 + D$	0	$D + D^2$	1	D^2	$1 + D^2$	$1 + D + D^2$
D^2	D^2	$1 + D^2$	$D + D^2$	0	$1 + D + D^2$	D	$1 + D$	1
$1 + D$	$1 + D$	D	1	$1 + D + D^2$	0	$1 + D^2$	D^2	$D + D^2$
$D + D^2$	$D + D^2$	$1 + D + D^2$	D^2	D	$1 + D^2$	0	1	$1 + D$
$1 + D + D^2$	$1 + D + D^2$	$D + D^2$	$1 + D^2$	$1 + D$	D^2	1	0	D
$1 + D^2$	$1 + D^2$	D^2	$1 + D + D^2$	1	$D + D^2$	$1 + D$	D	0

or with lsb last and octal notation, as in the above $GF(8)$-element table's last column.

$\oplus$	0	1	2	4	6	3	7	5
0	0	1	2	4	6	3	7	5
1	1	0	3	5	2	7	6	4
2	2	3	0	6	4	1	5	7
4	4	5	6	0	7	2	3	1
6	6	2	1	7	0	5	4	6
3	3	7	4	2	5	0	1	3
7	7	6	5	3	4	1	0	2
5	5	4	7	1	6	3	2	0

.

The multiplication tables require multiplication of the two polynomials and then substituting $D^3 = 1+D$, but can more simply determined by using the able list of elements and that $\alpha^i \cdot \alpha^j = \alpha^{i+j}$ (where $i+j$ is in $\mathbb{Z}$).

$\otimes$	0	1	D	D^2	$1+D$	$D+D^2$	$1+D+D^2$	$1+D^2$
0	0	0	0	0	0	0	0	0
1	0	1	D	D^2	$1+D$	$D+D^2$	$1+D+D^2$	$1+D^2$
D	0	D	D^2	$1+D$	$D+D^2$	$1+D+D^2$	$1+D^2$	1
D^2	0	D^2	$1+D$	$D+D^2$	$1+D+D^2$	$1+D^2$	1	D
$1+D$	0	$1+D$	$D+D^2$	$1+D+D^2$	$1+D^2$	1	D	D^2
$D+D^2$	0	$D+D^2$	$1+D+D^2$	$1+D^2$	1	D	D^2	$1+D$
$1+D+D^2$	0	$1+D+D^2$	$1+D^2$	1	D	D^2	$1+D$	$D+D^2$
$1+D^2$	0	$1+D^2$	1	D	D^2	$1+D$	$D+D^2$	$1+D+D^2$

.

Again using the lsb last notation, the table can be rewritten:

$\otimes$	0	1	2	4	6	3	7	5
0	0	0	0	0	0	0	0	0
1	0	1	2	4	6	3	7	5
2	0	2	4	6	3	7	5	1
4	0	4	6	3	7	5	1	2
6	0	6	3	7	5	1	2	4
3	0	3	7	5	1	2	4	6
7	0	7	5	1	2	4	6	3
5	0	5	1	2	4	6	3	7

.

Matlab has a finite field facility that allows simple execution of Galois Field arithmetic. For example the following sequence generates Figure B.1's multiplication table for $GF(5)$:

```
>> A=gf(ones(4,1)*[0 1 2 3],2)
 A = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
 Array elements =


           0           1           2           3
           0           1           2           3
           0           1           2           3
           0           1           2           3
>> B=gf([0 1 2 3]'*ones(1,4),2)
 B = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
 Array elements =
           0           0           0           0
           1           1           1           1
           2           2           2           2
           3           3           3           3

>> A.*B
 ans = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
 Array elements =
           0           0           0           0
           0           1           2           3
           0           2           3           1
           0           3           1           2
```

A slightly more complicated example is GF(16) with primitive polynomial $g(D) = D^4 + D + 1$ so (using hexadecimal notation)

$$(5 \cdot B)_{g(D)} = \left((D^2+1) \cdot (D^3+D+1)\right)_{D^4+D+1} \tag{B.15}$$

$$= \left(D^5 + D^3 + D^2 + D^3 + D + 1\right)_{g(D)} \tag{B.16}$$

$$= D \cdot (D+1) + D^2 + D + 1 = 1 \quad , \tag{B.17}$$

so $B = 5^{-1}$ in GF(16). A multiplication table for GF(16) can be found (using Matlab as):

```
A=gf(ones(16,1)*[0:15],4);
B=gf([0:15]'*ones(1,16),4);
A.*B
ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0
0  1  2  3  4  5   6   7   8   9  10  11  12  13  14  15
0  2  4  6  8 10  12  14   3   1   7   5  11   9  15  13
0  3  6  5 12 15  10   9  11   8  13  14   7   4   1   2
0  4  8 12  3  7  11  15   6   2  14  10   5   1  13   9
0  5 10 15  7  2  13   8  14  11   4   1   9  12   3   6
0  6 12 10 11 13   7   1   5   3   9  15  14   8   2   4
0  7 14  9 15  8   1   6  13  10   3   4   2   5  12  11
0  8  3 11  6 14   5  13  12   4  15   7  10   2   9   1
0  9  1  8  2 11   3  10   4  13   5  12   6  15   7  14
0 10  7 13 14  4   9   3  15   5   8   2   1  11   6  12
0 11  5 14 10  1  15   4   7  12   2   9  13   6   8   3
0 12 11  7  5  9  14   2  10   6   1  13  15   3   4   8
0 13  9  4  1 12   8   5   2  15  11   6   3  14  10   7
0 14 15  1 13  3   2  12   9   7   6   8   4  10  11   5
0 15 13  2  9  6   4  11   1  14  12   3   8   7   5  10
```

Alternately, enumeration of all 15 powers of a primitive element such as 2:

```
A=gf(gf((2*ones(16,1)'),4).^[0:15],4)
A = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)
1  2  4  8  3  6  12  11  5  10  7  14  15  13  9  1
\begin{verbatim}
where each element appears once until the element 1 repeats itself in the 16th position.  Further, t
\begin{verbatim}
gf(ones(1,16)./A,4)
1  9 13 15 14  7  10   5 11 12   6   3   8   4  2  1
```

which is simply the reversed sequence of powers of the primitive element. Thus, essentially 16 (4-bit) elements could be stored in 8 bytes of memory, adjoined with a high-speed 4-bit adder (for the index) along with simple 4-places of binary addition to create an exceptionally high-speed and low-cost $GF(16)$ general purpose arithmetic implementation. Even for $GF(256)$, only 256 bytes of storage and a simple 8-bit adder (plus eight 1-bit adders) are sufficient for high-speed very efficient arithmetic. When one of the elements is fixed, as is often the case in coding, much further simplification yet is possible, as in a later subsection.

Primitive elements of $GF(p^m)$ cannot be factored when viewed as polynomials with coefficients in $GF(p)$. Such primitive elements are however be roots of higher-degree polynomials with coefficients in $GF(p)$, for instance such a primitive element α always satisfies $f(D=\alpha)=\alpha^{p^m-1}-1=0$ and $f(D)$ in this case has coefficients (1 and -1) in $GF(p)$. The polynomial with $GF(p)$ coefficients of minimum degree for which $f_\alpha(D)=0$ when $D=\alpha$ is called the **minimal polynomial** of the primitive element α. Clearly $f_\alpha(D)$ must be a factor of $D^{p*m}-1$.

B.1.2.1 Conjugates

The polynomial $(\alpha+\beta)^p$ has binomial expansion

$$(\alpha+\beta)^p=\alpha^p+\left[\sum_{k=1}^{p-1}\binom{p}{k}\alpha*p-k\beta^k\right]+\beta^p \quad . \tag{B.18}$$

With mod p arithmetic where p is prime, then

$$\binom{p}{k}=p\cdot\frac{(p-1)\cdots(p-k+1)}{k!}=0 \bmod p \tag{B.19}$$

because this quantity is an integer and since $k < p$ cannot divide p, it must divide the remaining factors. Then the quantity is an integer multiple of p and thus zero. This then means that

$$(\alpha + \beta)^p = \alpha^p + \beta^p \tag{B.20}$$

for all prime p in modulo-p arithmetic, and thus for arithmetic in $GF(p^m)$ for any $m \geq 1$. The statement is obvious for $p = 2$ of course. Further, then it is true by induction that

$$(\alpha + \beta)^{p^i} = \alpha^{p^i} + \beta^{p^i} \tag{B.21}$$

since it holds for $i = 1$ already, and raising (B.21) to the p^{th} power corresponds to $i \rightarrow i + 1$ and then

$$(\alpha + \beta)^{p^{i+1}} = \left(\alpha^{p^i} + \beta^{p^i}\right)^p = \left(\alpha^{p^i}\right)^p + \left(\beta^{p^i}\right)^p = \alpha^{p^{i+1}} + \beta^{p^{i+1}} \quad . \tag{B.22}$$

The successive powers of a primitive element α^{p^i} for $i = 0, ...r - 1$ are called its **conjugates**. Since $GF(p^m)$ contains a finite number of elements, eventually this set must repeat and that occurs for the first r for which $\alpha^{p^r} = 1$. The set $\{\alpha, \alpha^p, ..., \alpha^{p^{r-1}}\}$ is called the conjugacy class of the primitive element α. Each primitive element has its own conjugacy class, and these classes are mutually exclusive (if not mutually exclusive one the two primitive elements has the other as a factor, a contradiction of their being prime elements).

All elements in a conjugacy class are roots of the corresponding class' minimal polynomial. This is easily proved by noting

$$\begin{aligned}
[f_\alpha(D = \alpha)]^{p^j} &= 0^p = 0 & \text{(B.23)}\\
&= \left[f_0 + f_1 \cdot \alpha + ... + f_j \cdot \alpha^j\right]^{p^j} & \text{(B.24)}\\
&= f_0^{p^j} + f_1^{p^j} \cdot \alpha^p + ... + f_j^{p^j} \cdot \alpha^{p^j} & \text{(B.25)}\\
&= f_0 + f_1 \cdot \alpha^p + ... + f_j \cdot \alpha^{p^j} & \text{(B.26)}\\
&= f_\alpha(\alpha^{p^j}) & \text{(B.27)}
\end{aligned}$$

so α^{p^j} is also a root. The minimal polynomial has minimum degree and contains all the elements in the conjugacy class so that degree is equal to the number of elements in the conjugacy class, namely

$$f_\alpha(D) = (D - \alpha) \cdot (D - \alpha^p) \cdot .... \cdot (D - \alpha^{r-1}) \;\; l. \tag{B.28}$$

There is a minimal polynomial for each of the primitive elements (and all products of primitive elements, generating thus the entire non-zero portion of the Galois Field). Since all the non-zero elements are determined by

$$D^{p^m-1} - 1 = 0 \tag{B.29}$$

then

$$D^{p^m-1} - 1 = (D - 1) \cdot \prod_\alpha f_\alpha(D) \quad . \tag{B.30}$$

B.2 Lattices

The theory of coset codes depends heavily on the concept of a lattice:

> **Definition B.2.1 (Lattice)** *A* **lattice**, Λ, *is an N-dimensional group of points that is closed under addition in that the sum of any two points in the group is also a point in the group. Lattice addition is presumed to be real vector addition where used in this text.*
>
> *Any constellation that is a subset of Λ is also denoted by Λ (in a slight abuse of notation), and the number of constellation points in such a Λ is written $|\Lambda|$.*

Examples of lattices include, Z, the (one-dimensional) set of integers; Z^2, the two-dimensional set of all ordered-pairs of any two integers, $Z^2 \stackrel{\Delta}{=} \{(x_1, x_2) | x_1 \in Z,\ x_2 \in Z\}$; Z^N, the N-dimensional integer lattice, and D_2, a two-dimensional lattice that is formed by taking "every other point" from Z^2 (that is take the points where $x_1 + x_2$ is a even integer). Λ' is a sublattice of Λ, where a sublattice is defined as

> **Definition B.2.2 (Sublattice)** *A* **sublattice** Λ' *of a lattice Λ is an N-dimensional lattice of points such that each point is also a point in Λ.*

> **Definition B.2.3 (Coset of a Lattice)** *A* **coset of a lattice** *is an N-dimensional set of points, written $\Lambda + \boldsymbol{c}$, described by the translation*
>
> $$\Lambda + \boldsymbol{c} \stackrel{\Delta}{=} \left\{\boldsymbol{x} \mid \boldsymbol{x}' + \boldsymbol{c} \ ; \ \boldsymbol{x}' \in \Lambda \ , \ \boldsymbol{c} \in \mathcal{R}^N\right\} \quad , \tag{B.31}$$
>
> *where $\mathcal{R}^N$ is the N-dimensional set of vectors with real components.*

A sublattice Λ' **partitions** its parent lattice Λ into a group of cosets of Λ' whose union is λ. The partitioning is written Λ/Λ and the set of all cosets as $[\Lambda/\Lambda]$. The number of subsets is called the **order of the partition**, $|\Lambda/\Lambda'|$ and thus

$$\Lambda = \{\lambda + \boldsymbol{c} | \lambda \in \Lambda' \ , \ \boldsymbol{c} \in [\Lambda/\Lambda]\} \quad . \tag{B.32}$$

A **partition chain** is formed by further partitioning of the sublattice into its sublattices and can be abbreviated:

$$\Lambda/\Lambda'/\Lambda'' \quad . \tag{B.33}$$

As a simple example of partitioning in two dimensions

$$Z^2/D_2/2Z^2/2D_2/4Z^4 ... \tag{B.34}$$

Each subsequent partition has order 2, that is there are two cosets of the sublattice to form the immediate parent lattice. Also, a four-way partition would be $Z^2/2Z^4$ which has $|Z^2/2Z^4| = 4$ and $\left[Z^2/2Z^4\right] = \{(0,0),\ (1,0),\ (0,1),\ (1,1)\}$. More sophisticated lattices in two and higher dimensions are introduced and used in Sections B.5 and code6.

B.2.1 Elementary Lattice Operations

The **cartesian product** of one lattice with another has dimensionality equal to the sum of the dimensionality of the original two lattices and is formed (as was defined in Chapter 1) by taking all possible points in the first lattice and pairing them with each and every point in the second lattice. This is typically written

$$\Lambda_1 \otimes \Lambda_2 = \{(\lambda_1, \lambda_2) \mid \lambda_1 \in \Lambda_1 \quad , \quad \lambda_2 \in \Lambda_2\} \quad . \tag{B.35}$$

An important special case of the cartesian product for lattices is the so-called squaring construction:

> **Definition B.2.4 (Squaring Construction)** *The* **squaring construction**, *performed on a lattice Λ, is given by*
>
> $$\Lambda^2 \stackrel{\Delta}{=} \Lambda \otimes \Lambda \quad , \tag{B.36}$$
>
> *that is, the (cartesian) product of Λ with itself.*

Examples of the squaring construction are $Z^2 = Z \otimes Z$, $Z^4 = Z^2 \otimes Z^2$, and $Z^8 = Z^4 \otimes Z^4$. The cartesian product has a "distributive" property over set union:

$$\left(A \bigcup B\right) \otimes \left(C \bigcup D\right) = (A \otimes C) \bigcup (A \otimes D) \bigcup (B \otimes C) \bigcup (B \otimes D) \quad . \tag{B.37}$$

The **Rotation Operator** R_N is used to rotate a lattice when N is even.

Definition B.2.5 (Rotation Operator) *The rotation operator R_2 is defined by*

$$R_2 \triangleq \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{B.38}$$

It is applied to a two-dimensional lattice Λ in the following sense:

$$R_2\Lambda = \left\{ (x, y) \mid \begin{bmatrix} x \\ y \end{bmatrix} = R_2 \begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} \ni (\lambda_x, \lambda_y) \in \Lambda \right\} \quad . \tag{B.39}$$

A larger-dimensional rotation matrix is computed recursively according to

$$R_{2N} = \begin{bmatrix} R_N & 0 \\ 0 & R_N \end{bmatrix} \quad . \tag{B.40}$$

B.2.2 Binary Lattices and Codes

Binary lattices characterize most used coset codes. A lattice $\Lambda_(N, k)$ is a **binary lattice** if it can be partioned by the lattice $2Z^N$. In other words, a partition chain

$$Z^N / \Lambda_{(N,k)} / 2Z^N \tag{B.41}$$

exists for a binary lattice. The overall partition $Z^N / 2Z^N$ clearly has order $|Z^N / 2Z^N| = 2^N$. When the subscript (N, k) is used, then

$$|Z^N / \Lambda_{(N,k)}| \quad = \quad 2^{N-k} = 2^{r_G} \tag{B.42}$$

$$|\Lambda_{(N,k)} / 2Z^N| \quad = \quad 2^k \tag{B.43}$$

so that $r_G \triangleq N - k$. The number r_G is associated with the "number of parity bits" of a related binary code, just as r_G is used in Section B.5 to characterize the number of parity bits in underlying convolutional codes. A binary lattice may be mapped to a binary code C with generator G and r_G parity bits so that N-dimensional binary codewords $\boldsymbol{v}$ are generated from any binary input k-tuple $\boldsymbol{u}$ according to

$$\boldsymbol{v} = \boldsymbol{u}G \quad . \tag{B.44}$$

Any point in the corresponding binary lattice, $\boldsymbol{\lambda}$, can be constructed as

$$\boldsymbol{\lambda} = \boldsymbol{v} + 2Z^N \quad . \tag{B.45}$$

Essentially then the 2^k distinct codewords of the binary code C are the set of coset leaders in the partition of the binary lattice $\Lambda_{(N,k)}$ by $2Z^N$. This may also be written as

$$\boldsymbol{\lambda} = \boldsymbol{v} + 2\boldsymbol{m} \quad , \tag{B.46}$$

where $\boldsymbol{m} \in Z^N$. The mapping to a binary code can provide a simple mechanism to generate points in the binary lattice. As discussed in Section **??**, a **dual code** for the binary code is defined by a parity matrix H for the original code. The parity matrix H has the property for any codeword $\boldsymbol{v}$ in C that

$$\boldsymbol{v}H^* = 0 \quad , \tag{B.47}$$

or the rows of the parity matrix are orthogonal to the codewords. The dimensionality of the matrix H is $(N-k) \times N = r_G \times N$ and defines the dual code $C^{\perp}$ such dual-code codewords are generated by any $(N-k)$ dimensional binary vector according to

$$\boldsymbol{v}^{\perp} = \boldsymbol{u} H^* \quad . \tag{B.48}$$

Any codeword in the dual code is orthogonal to all codewords in the original code and vice-versa:

$$\boldsymbol{v}^{\perp} \boldsymbol{v}^* = 0 \quad . \tag{B.49}$$

The dual code of a linear binary code is clearly also a linear binary code[1] and thus defines a binary lattice itself. This binary lattice is known as the **dual lattice** $\Lambda^{\perp}_{(N,k}$. There is a partition chain

$$Z^N / \Lambda^{\perp}_{(N,k} / 2Z^N \quad , \tag{B.50}$$

with

$$\begin{aligned} |Z^N / \Lambda^{\perp}_{(N,k}| &= 2^k & \text{(B.51)} \\ |\Lambda^{\perp}_{(N,k} / 2Z^N| &= 2^{N-k} \quad . & \text{(B.52)} \end{aligned}$$

Then, the inner product of any $\boldsymbol{\lambda}$ point in the original lattice is orthogonal modulo-2 to any point in the dual lattice

$$\boldsymbol{\lambda}^{\perp} \boldsymbol{\lambda}^* = \boldsymbol{v}^{\perp} \boldsymbol{v}^* + 2 \cdot (\text{some integer vector}) \tag{B.53}$$

or thus

$$\left(\boldsymbol{\lambda}^{\perp} \boldsymbol{\lambda}^* \right)_2 = 0 \quad . \tag{B.54}$$

The notation $(\cdot)_2$ means modulo-2 on all components (so even integers go to zero and odd integers go to 1). The dual lattice has 2^{r_G} cosets in $2Z^N$. Decoding of any binary N-vector $\tilde{\boldsymbol{v}}$ for the closest vector in the code C often computes an $r_G = (N-k)$-dimensional vector **syndrome** $\boldsymbol{s}$ of $\tilde{\boldsymbol{v}}$ as

$$\boldsymbol{s} \stackrel{\Delta}{=} \tilde{\boldsymbol{v}} H^* \quad . \tag{B.55}$$

The syndrome $\boldsymbol{s}$ can be any of the 2^{r_G} possible r_G-dimensional binary vectors since the rank of H is r_G and $\tilde{\boldsymbol{v}}$ can be any binary vector. The syndrome concept can be generalized to the lattice $\Lambda_{(N,k)}$ for any N-dimensional vector in Z^N because

$$\boldsymbol{s} = \left(\tilde{\boldsymbol{\lambda}} H^* \right)_2 \quad . \tag{B.56}$$

An example of such a binary lattice in two dimensions is D_2, since $Z^2/D_2/2Z^2$. The corresponding binary code is a rate 1/2 code with $r_G = 1$ and codewords $\{(0,0),\ (1,1)\}$. This code is its own dual. The $2^{r_G} = 2$ syndromes are the one-dimensional vectors $\boldsymbol{s} \in \{0,1\}$. These two one-dimensional syndromes can be found by taking any two-dimensional binary vector $\tilde{\boldsymbol{v}} \in \{(0,0),\ (0,1),\ (1,0),\ (1,1)\}$ and multiplying by $H^* = [1\ 1]^*$.

Any point in a coset of the partition set $\left[Z^N / \Lambda_{(N,k)} \right]$ can again be written as

$$\tilde{\boldsymbol{\lambda}} = \boldsymbol{c} + \boldsymbol{\lambda} \tag{B.57}$$

where $\boldsymbol{c} \in Z^2$. Thus, any "received" point in the same coset of $\Lambda_{(N,k)}$ will have the same syndrome, which is easily proven by writing

$$\boldsymbol{s} = (\boldsymbol{c} + \boldsymbol{\lambda}) H^* = (\boldsymbol{c} + \boldsymbol{v} + 2\boldsymbol{m}) H^* = (\boldsymbol{c} H^*)_2 \quad . \tag{B.58}$$

Thus, the 2^{r_G} cosets of $\Lambda_{(N,k)}$ in Z^N are enumerated by any and all of the 2^{r_G} possible distinct r_G-dimensinal binary syndrome vectors of the code C, which are simply all possible r_G-dimensional binary vectors.

[1] If $\boldsymbol{v}_1^{\perp}$ and $\boldsymbol{v}_2^{\perp}$ are in $C^{\perp}$, then $\left(\boldsymbol{v}_1^{\perp} + \boldsymbol{v}_2^{\perp} \right) \boldsymbol{v}^* = 0$ and thus $\left(\boldsymbol{v}_1^{\perp} + \boldsymbol{v}_2^{\perp} \right)$ is also in $C^{\perp}$.

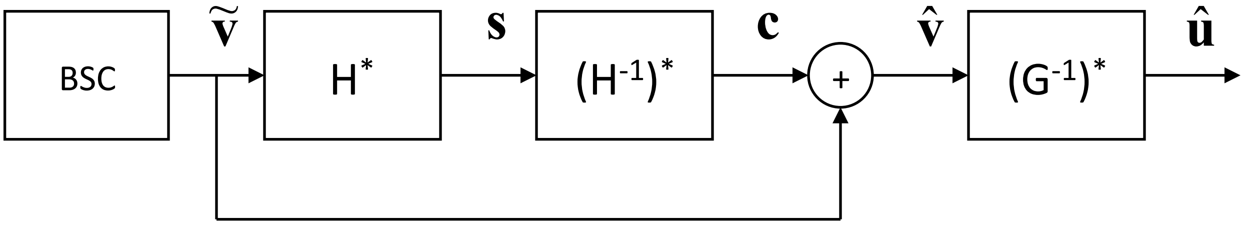


Figure B.3: Basic syndrome decoder.

The r_G-dimensional parity matrix H has a right inverse H^{-1} such that (binary operations implied)

$$HH^{-1} = I_{r_G} \quad . \tag{B.59}$$

Similarly by taking the transpose

$$(H^{-1})^* H^* = I_{r_G} \quad . \tag{B.60}$$

Thus the 2^{r_G} N-dimensional coset leaders can be enumerated according to

$$\boldsymbol{c} = \left(\boldsymbol{s}(H^{-1})^*\right)_2 \quad , \tag{B.61}$$

where $\boldsymbol{s}$ runs through all 2^{r_G} possible binary r_G-dimensional vectors. The syndromes correspond to the 2^{r_G} possible error vectors or offsets from actual transmitted codewords on a binary channel. After computing the syndrome, the corresponding binary vector $\boldsymbol{c}$ that satisfies $\boldsymbol{s} = \tilde{\boldsymbol{v}}H^*$ can be computed by Equation (B.61, then leading to $\hat{\boldsymbol{v}} = \tilde{\boldsymbol{v}} \oplus \boldsymbol{c}$. If the channel is a BSC, then $\hat{\boldsymbol{v}}$ is the ML estimate of the codeword (and indeed if the code is systematic, then the input is determined). For non-systematic codes, the relationship $\hat{\boldsymbol{u}} = \hat{\boldsymbol{v}}\left(G^{-1}\right)^*$ determines the ML estimate of the input on the BSC. Figure B.3 illustrates such binary block-code decoding. Syndromes are used in the shaping codes of Section **??** and in binary block codes of Section **??**.

B.2.2.1 Association of lattices with binary codes

Partitionings of lattices can be associated with various binary codes, some of which are well known. This subsection lists some of those partitionings and associated codes. To describe a binary code, it will be indicated by an ordered triple (N, k, d_{free}). Thus, for instance, (4,3,2) would describe a binary code with $2^3 = 8$ codewords of length 4 bits each and a minimum Hamming distance between the closest 2 of 2 bit positions. A code with (N,k=N,1) must necessarily be uncoded and have free distance 1. A code with (N,0, ∞) has infinite free distance and only one codeword of all zeros.

One-dimensional partitioning of binary lattices The one-dimensional partitioning chain

$$Z/2Z \tag{B.62}$$

is somewhat trivial and corresponds to the code (1,1,1).

Two-dimensional partitioning of binary lattices: The two-dimensional partitioning chain of interest is

$$Z^2/D_2/2Z^2 \tag{B.63}$$

making D_2 a binary lattice associated with the code (2,1,2). The codewords of this code are [0 0] and [1 1]. The lattice D_2 can be written as

$$D_2 = 2Z^2 + u_1 \cdot \underbrace{[1\ 1]}_{G_{(2,1,2)}} \tag{B.64}$$

where u_1 is the single input bit to the rate-1/2 linear binary code with generator $G_{D_2} = G_{(2,1,2)} = [1\ 1]$. Such a binary code, and thus the associated lattice, is its own dual so

$$H_{D_2} = G_{D_2} = G_{D_2^{\perp}} \tag{B.65}$$

and

$$H^{-*} = [0\ 1] \tag{B.66}$$

is an acceptable left inverse for H^*.

Four-dimensional parititioning of binary lattices The four-dimensional partitioning chain of most interest in code is

$$Z^4/D_4/R_4Z^4/R_4D_4/2Z^4 \quad , \tag{B.67}$$

which is equivalent to the linear binary code partitioning chaing (each linear code is a linear binary-code subset of its parent in the chain)

$$\underbrace{(4,4,1)}_{Z^4} / \underbrace{(4,3,2)}_{D_4} / \underbrace{(4,2,2)}_{R_4Z^4} / \underbrace{(4,1,4)}_{R_4D_4} / \underbrace{(4,0,\infty)}_{2Z^4} \quad . \tag{B.68}$$

The generating matrices can easily be built from the bottom of the chain upward:

$$R_4D_4 \quad = \quad 2Z^4 + u_1 \cdot \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,1,4)} \tag{B.69}$$

$$R_4Z^4 \quad = \quad 2Z^4 + [u_2\ u_1] \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,2,2)} \tag{B.70}$$

$$D_4 \quad = \quad 2Z^4 + [u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,3,2)} \tag{B.71}$$

$$Z^4 \quad = \quad 2Z^4 + [u_4\ u_3\ u_2\ u_1] \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{B.72}$$

Each successive step adds another row to the generator for the partitioning. The dual codes also form a "reverse" partition chain

$$\underbrace{(4,4,1)}_{Z^4} / \underbrace{(4,3,2)}_{(R_4D_4})^{\perp} / \underbrace{(4,2,2)}_{R_4Z^4} / \underbrace{(4,1,4)}_{D_4^{\perp}} / \underbrace{(4,0,\infty)}_{2Z^4} \quad . \tag{B.73}$$

The lattice $R_4D_4 = (D_2)^2$ is a self-dual and so its generator and parity matrix are the same. However,

$$D_4^{\perp} \quad = \quad R_4D_4 \tag{B.74}$$

$$(R_4D_4)^{\perp} \quad = \quad D_4 \tag{B.75}$$

and thus

$$H_{D_4} \quad = \quad G_{R_4D_4} \tag{B.76}$$

$$H_{R_4D_4} \quad = \quad G_{D_4} \quad . \tag{B.77}$$

Eight-dimensional parititioning of binary lattices: The eight-dimensional partitioning chain of most interest in code is

$$Z^8/D_8/(D_4)^2/DE_8/E_8/R_8D_8/(R_4D_4)^2/R_8DE_8/2Z_8 \quad , \tag{B.78}$$

which is equivalent to the linear binary code partitioning chaing (each linear code is a linear binary-code subset of its parent in the chain)

$$\underbrace{(8,8,1)}_{Z^8}/\underbrace{(8,7,2)}_{D_8}/\underbrace{(8,6,2)}_{(D_4)^2}/\underbrace{(8,5,2)}_{DE_8}/\underbrace{(8,4,4)}_{E_8}/\underbrace{(8,3,4)}_{R_8D_8}/\underbrace{(8,2,4)}_{(R_4D_4)^2}/\underbrace{(8,1,4)}_{R_8DE_8}/\underbrace{(8,0,\infty)}_{Z^8} \quad . \tag{B.79}$$

The generating matrices can easily be built from the bottom of the chain upward, starting this time with one generator for E_8 and working upward:

$$E_8 \quad = \quad Z^8 + [u_4\ u_3\ u_2\ u_1] + \underbrace{\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}}_{(8,4,4)} \tag{B.80}$$

$$DE_8 \quad = \quad Z^8 + [u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}}_{(8,5,2)} \tag{B.81}$$

$$(D_4)^2 \quad = \quad Z^8 + [u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}}_{(8,6,2)} \tag{B.82}$$

$$D_8 \quad = \quad Z^8 + [u_7\ u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}}_{(8,7,2)} \tag{B.83}$$

$$D_8 \quad = \quad Z^8 + [u_8\ u_7\ u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}}_{(8,8,1)} \tag{B.84}$$

These generators are not unique for any of the binary codes, but each provides a partitioning chain that corresponds to the sequence of binary codes shown, or more generally to binary linear codes with the correct parameters for a valid lattice partitioning chain. Thus, the rows need not match exactly those in the partitioning sections for multi-dimensional trellis codes and lattices. The Gosset lattice is its own dual. Also, R_8DE_8 is the dual of D_8, $(R_4D_4)^2$ would already have been known from four-dimensional partitioning to be the dual of $(D_4)^2$. Finally, R_8D_8 is the dual of DE_8.

B.2.2.2 16, 24, and 32 dimensional partitioning chains

Partitions in 16, 24, and 32 dimensions are sometimes used in advanced coset coding, but not in this chapter. They may be binary lattices or mod-4 lattices (meaning they have $4Z^N$ as a sub-lattice). However, it is possible to associate binary lattice partition chains with the code partitioning

$$(16,16,1)/(16,15,2)/..../(16,1,16)/(16,0,\infty) \quad . \tag{B.85}$$

Some special lattices sometimes used in coding are

$$D_{16} \triangleq 2Z^{16} + (16,15,2) \tag{B.86}$$
$$H_{16} \triangleq 2Z^{16} + (16,11,4) \tag{B.87}$$
$$\Lambda_{16} \triangleq 4Z^{16} + 2(16,15,2) + (16,5,8) \tag{B.88}$$

The lattices H_{16} (H for "half" lattice) and Λ_{16} are sometimes called 16-dimensional "Barnes-Wall" lattices and Λ_{16} has a coding gain 4.52 dB ($2^{1.5}$), while H_{16} has a coding gain of $2^{\frac{11}{8}}$=4.14 dB. D_{16} is a 16-dimensional checkerboard and has coding gain of $2^{7/8} = 2.63$ dB.

32 dimensional lattices follow the same binary linear code partitioning (now with 32 steps) with

$$D_{32} \triangleq 2Z^{32} + (32,31,2) \tag{B.89}$$
$$X_{16} \triangleq 2Z^{32} + (32,26,4) \tag{B.90}$$
$$H_{32} \triangleq 4Z^{32} + 2(32,31,2) + (32,16,8) \tag{B.91}$$
$$\Lambda_{32} \triangleq 4Z^{32} + 2(32,26,4) + (32,6,16) \quad . \tag{B.92}$$

These are all also (32-dimensional) Barnes-Wall lattices wit coding gains $2^{\frac{15}{16}}$=2.82 dB, $2^{\frac{13}{8}}$=4.89 dB, $2^{\frac{31}{16}}$=5.83 dB, and 4=6.02, dB respectively.

There is also a 24-dimensional series of partitions in the same fashion that is of particular interest because it contains a very special high-gain lattice Λ_{24} known as the Leech lattice

$$D_{24} \triangleq 2Z^{24} + (24,23,2) \tag{B.93}$$
$$X_{24} \triangleq 2Z^{24} + (24,18,2) \tag{B.94}$$
$$H_{24} \triangleq 4Z^{32} + 2(24,23,2) + (24,12,8) \tag{B.95}$$
$$\Lambda_{32} \triangleq 4Z^{24} + 2(24,18,4) + (24,6,16)' \quad . \tag{B.96}$$

the notation (24,6,16)' means the set of all binary linear combinations modulo 4 of a set of six generators wose coordinates are integers modulo 4. This not a binary lattice, but what is called a mod-4 lattice. These have coding gains $2^{\frac{11}{12}}$=2.76 dB, $2^{\frac{3}{2}}$=4.52 dB, $2^{\frac{23}{12}}$=5.77 dB, and 4=6.02, dB respectively.

B.3 Coset Codes, Lattices, and Partitions

The general coset-code encoder is shown in Figure B.4. This encoder consists of 3 major components: the **binary encoder** (G), the **coset select** (CS), and the **signal select** (SS). The encoder output, $\boldsymbol{x}_m$, is an N-dimensional vector sequence of points; m is a symbol-time index. Each (N-dimensional) symbol of this sequence is chosen from an N-dimensional constellation. The sequences of $\boldsymbol{x}_m$ are the codewords $\boldsymbol{x}(D) = \sum_m \boldsymbol{x}_m D^m$. This signal constellation consists of 2^{b+r_G} signal points in some coset of an N-dimensional real lattice, Λ (for a definition of a lattice and its cosets, see Appendix B.2). The basic idea in designing a coset code is to select carefully N-dimensional sequences that are separated by a large minimum distance. The signal constellation contains 2^{k+r_G} subsets (**cosets**) of the constellation that each contain 2^{b-k} points. A good trellis code is designed by carefully selecting sequences of cosets that will lead to the desired increase in separation. The vector sequence $\boldsymbol{x}_m$ is converted by modulation to a continuous-time waveform according to the techniques discussed in Chapter 1.

At any time m, there are b input bits to the encoder, of which k are input to a conventional binary encoder (convolutional or block), which produces $n = k + r_G$ output bits, with r_G specifying the number of redundant bits produced by the encoder ($r_G = n - k$ in Sections **??** and **??**). The quantity $\bar{r}_G \triangleq r_G/N$ is called the **normalized redundancy** of the convolutional encoder in the coset-code encoder. The quantity $\bar{k}_G \triangleq k/N$ is called the **informativity** of the convolutional encoder in the coset-code encoder. The $k + r_G$ output bits of the binary encoder specify one of 2^{k+r_G} disjoint subsets (or cosets) into which the signal constellation has been divided. This subset selection is performed by the coset select. The current output signal point $\boldsymbol{x}_m$ at time m is selected by the signal select from the 2^{b-k} remaining points in that coset that is currently selected by the coset select. The set of all possible sequences that can be generated by the coset encoder is called the **coset code** and is denoted by C.

In Figure B.4, the sublattice Λ' is presumed to partition the lattice Λ, written $\Lambda|\Lambda'$, into 2^{k+r_G} cosets of Λ', where each coset has all of its points contained within the original lattice Λ. Such a partitioning always accompanies the specification of a coset code.

> **Definition B.3.1 (Coset Partitioning $\Lambda|\Lambda'$)** *A* **coset partitioning** *is a partition of the Lattice Λ into $|\Lambda|\Lambda'|$ (called the "order" of the partition) cosets of a sublattice Λ' such that each point in the original lattice Λ is contained in one, and only one, coset of the sublattice Λ'.*

The coset select in Figure B.4 then accepts the $k+r_G$ bits from G as input and outputs a corresponding index specifying one of the cosets of Λ' in the coset partitioning $\Lambda|\Lambda'$. There are 2^{b+r_G} constellation points chosen from the lattice Λ, and each is in one of the 2^{k+r_G} cosets. There are an equal number, 2^{b-k}, of constellation points in each coset. The signal select accepts $b-k$ uncoded input bits and selects which of the 2^{b-k} points in the coset of Λ' specified by the coset select that will be transmitted as the modulated vector $\boldsymbol{x}_m$.

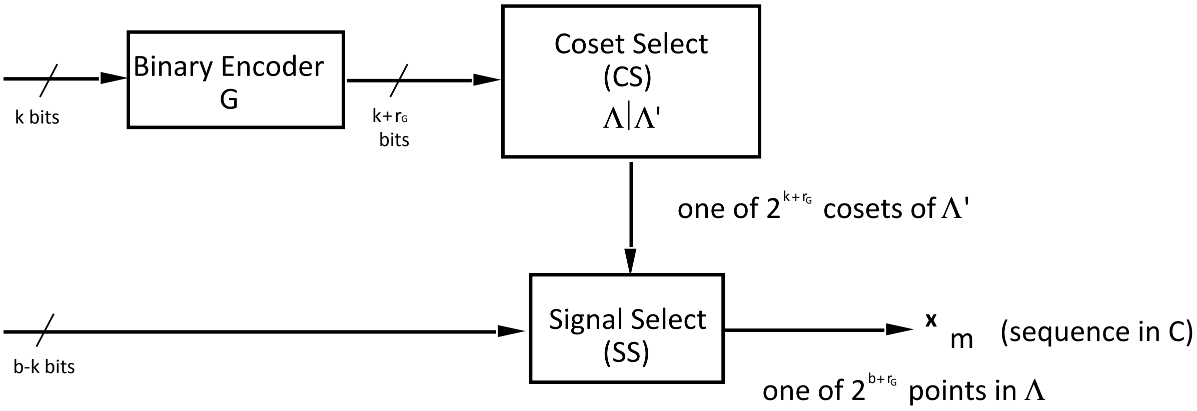


Figure B.4: The coset-code encoder

If the encoder G is a convolutional encoder, then the set of all possible transmitted sequences $\{\boldsymbol{x}(D)\}$ is a **Trellis Code**, and if G is a block encoder, the set of N-dimensional vectors is a **Lattice Code**. So, both trellis codes and lattice codes are coset codes.

B.3.1 Gain of Coset Codes

There are several important concepts in evaluating the performance of a coset code, but the gain is initially the most important.

The fundamental gain will be taken in Volume II to always be with respect to a uncoded system, $\tilde{\boldsymbol{x}}$, that uses points on the N-dimensional integer lattice (denoted Z^N). It is easy (and often harmless, but not always) to forget the difference between $\mathcal{V}_{\boldsymbol{x}}$ and $\mathcal{V}^{2/N}(\Lambda)$. $\mathcal{V}_{\boldsymbol{x}}$ is the volume of the constellation and is equal to the number of points in the constellation times $\mathcal{V}^{2/N}(\Lambda_{\boldsymbol{x}})$. For coding gain calculations where the two compared systems have the same number of points, this difference is inconsequential. However, in trellis codes, the two are different because the coded system often has extra redundant points in its constellation. For the uncoded reference Z^N lattice, $\mathcal{V}(Z^N) = 1$ and $d_{\text{min}}(Z^N) = 1$, so that the fundamental gain of a coset code reduces to

$$\gamma_f = \frac{\frac{d^2_{\text{min}}(\boldsymbol{x})}{\mathcal{V}^{2/N}_{\boldsymbol{x}}}}{\frac{d^2_{\text{min}}(\tilde{\boldsymbol{x}})}{\tilde{\mathcal{V}_{\boldsymbol{x}}}^{2/N}}} = \frac{\frac{d^2_{\text{min}}(\boldsymbol{x})}{2^{2(\bar{b}+\bar{r}_G)}\mathcal{V}^{2/N}(\Lambda)}}{\frac{d^2_{\text{min}}(\tilde{\boldsymbol{x}})}{2^{2\bar{b}}\cdot\mathcal{V}^{2/N}(\Lambda)}} = \frac{\frac{d^2_{\text{min}}(C)}{2^{2\bar{r}_G}\cdot\mathcal{V}^{2/N}(\Lambda)}}{\frac{1}{1}} = \frac{d^2_{\text{min}}(C)}{\mathcal{V}(\Lambda)^{2/N}2^{2\bar{r}_G}} \tag{B.97}$$

The quantity $\bar{r}_G = \frac{r_G}{N}$ is the normalized redundancy of the encoder G. The gain γ_f in dB is

$$\begin{aligned}\gamma_f &= 10\cdot\log_{10}\left(\frac{d^2_{\text{min}}(C)}{\mathcal{V}(\Lambda)^{2/N}2^{2\bar{r}_G}}\right) && \text{(B.98)}\\ &= 20\log_{10}\left(\frac{d_{\text{min}}(C)}{\mathcal{V}(\Lambda)^{1/N}2^{\bar{r}_G}}\right) && \text{(B.99)}\end{aligned}$$

The redundancy of the overall coset code requires the concept of the redundancy of the original lattice Λ.

Definition B.3.2 (Redundancy of a Lattice) *The* **redundancy of a lattice** *is defined by r_Λ such that*

$$\mathcal{V}(\Lambda) = 2^{r_\Lambda} = 2^{N\bar{r}_\Lambda} \quad , \tag{B.100}$$

or $r_\Lambda = \log_2(\mathcal{V}(\Lambda))$ bits per symbol. The quantity $\bar{r}_\Lambda = r_\Lambda/N$ is called the **normalized redundancy** *of the lattice, which is measured in bits/dimension.*

Then, the fundamental coding gain of a coset code is

$$\gamma_f = \frac{d^2_{\text{min}}(C)}{2^{2(\bar{r}_G+\bar{r}_\Lambda)}} = \frac{d^2_{\text{min}}(C)}{2^{2\bar{r}_C}} \tag{B.101}$$

where

$$\bar{r}_C = \bar{r}_G + \bar{r}_\Lambda \quad . \tag{B.102}$$

Equation (B.101) is often used as a measure of performance in evaluating the performance of a given trellis code. Good coset codes typically have 3 dB $\leq \gamma_f \leq$ 6 dB.

Shaping gain is also important in evaluating trellis codes, but is really a function of the shape of the constellation used rather than the spacing of sequences of lattice points. The shaping gain is defined as (again comparing against a zero-mean translate of Z^N lattice)

$$\gamma_s = \frac{\mathcal{V}^{2/N}(\Lambda)\cdot 2^{2\bar{r}_G}}{\bar{\mathcal{E}}(\Lambda)} / \frac{1}{(2^{2\bar{b}}-1)/12} = \frac{2^{2\bar{r}_C}}{12\bar{\mathcal{E}}}(2^{2\bar{b}}-1) \quad , \tag{B.103}$$

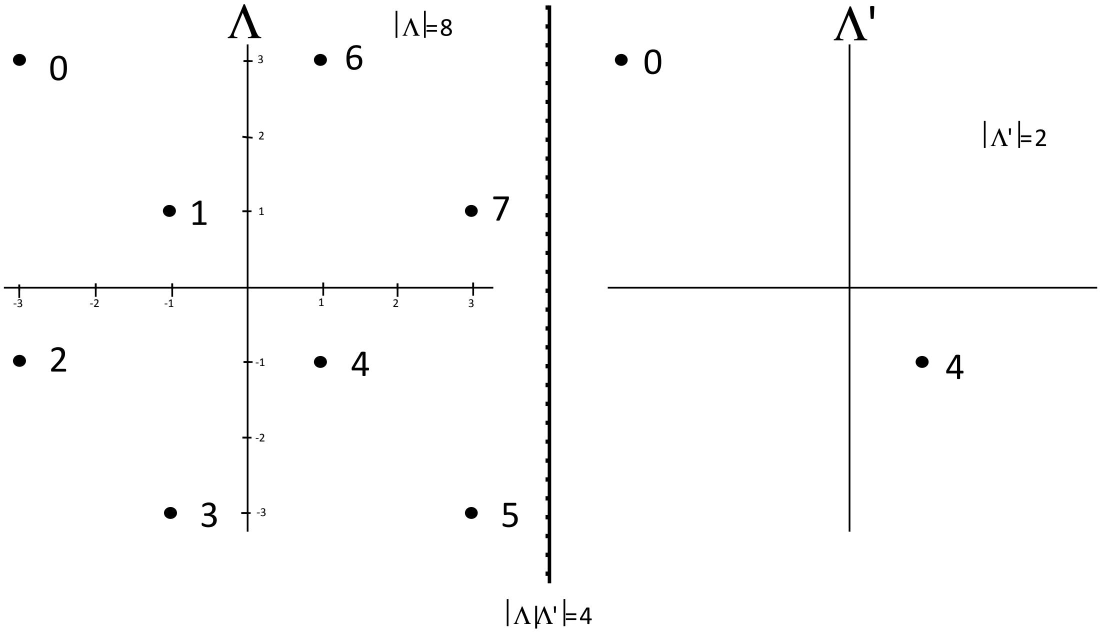


Figure B.5: Partitioning (a coset of) the D_2 Lattice

where $\bar{\mathcal{E}}(C)$ is the energy per dimension required for the coset code. Using the so-called "continuous approximation" (which holds accurately for $\bar{b} \geq 3$), the shaping gain is often approximated by

$$\gamma_s \approx \frac{2^{2\bar{r}_C} \cdot 2^{2\bar{b}}}{12\bar{\mathcal{E}}(\Lambda)} \quad . \tag{B.104}$$

This can also be written in dB as

$$\gamma_s \approx 10\log_{10}\left(\frac{\mathcal{V}^{2/N}(\Lambda)2^{2(\bar{b}+\bar{r}_G)}}{12\bar{\mathcal{E}}(\Lambda)}\right) \tag{B.105}$$

$$= 10\log_{10}\left(\frac{2^{2(\bar{b}+\bar{r}_C)}}{12\bar{\mathcal{E}}(\Lambda)}\right) \tag{B.106}$$

This shaping gain has a theoretical maximum of 1.53dB - the best known shaping methods achieve about 1.1dB (see Section **??**).

EXAMPLE B.3.1 (Ungerboeck's Rate 1/2 3 dB Trellis Code) In Figure B.5, the 8AMPM (or 8CR) constellation is subset of a (possibly scaled and translated) version of what is known as the $\Lambda = D_2$ Lattice that contains $|\Lambda| = 8$ points. The average energy per symbol for this system is $\mathcal{E} = 10$ or $\bar{\mathcal{E}} = 5$. The (translated) sublattice Λ' has a coset, Λ_0 that contains $|\Lambda_0| = 2$ points, so that there are $|\Lambda|\Lambda'| = 4$ cosets of Λ' in Λ; they are $\Lambda_0 = \{0, 4\}$, $\Lambda_1 = \{1, 5\}$, $\Lambda_2 = \{2, 6\}$, and $\Lambda_3 = \{3, 7\}$. These 4 cosets are selected by the two bit output $\boldsymbol{v}(D)$ of a rate 1/2 convolutional encoder with generator:

$$G(D) = \begin{bmatrix} 1 + D^2 & D \end{bmatrix} \quad . \tag{B.107}$$

The corresponding trellis and trellis encoder are shown in Figures B.6 and B.7, respectively. The convolutional encoder, G, has rate $r = 1/2$, but the overall coset-code has $\bar{b} = 2$ bits/2dimensions = 1 bit/dimension. The input bit $u_{1,k}$ passes through G, where the two

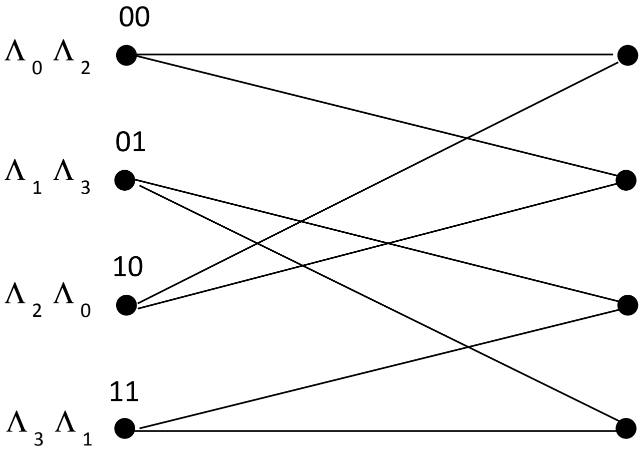


Figure B.6: Trellis for 4-state rate 1/2 Ungerboeck Code

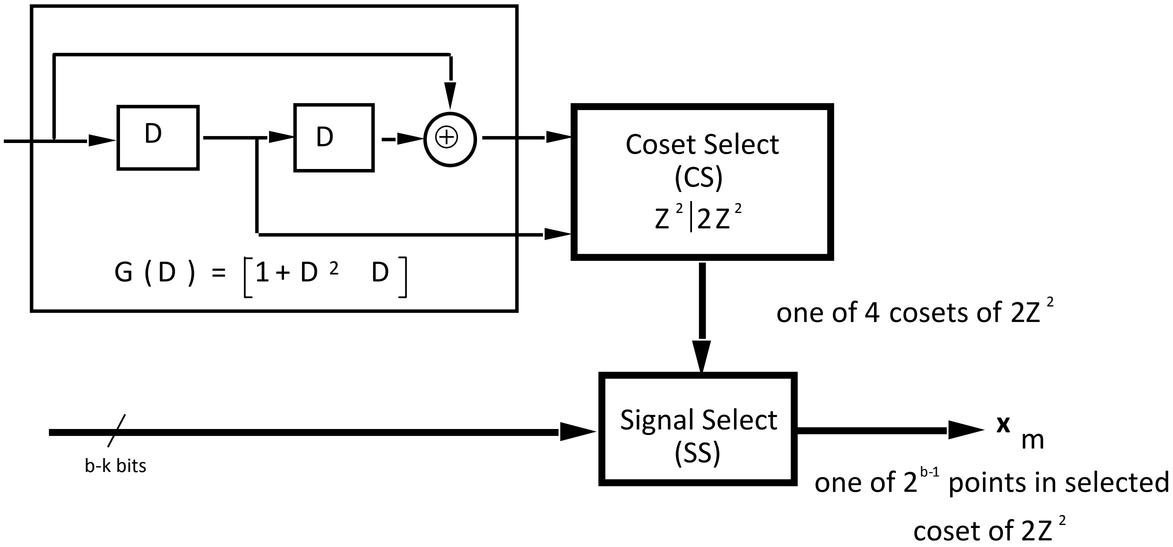


Figure B.7: Encoder for 4-state rate 1/2 Ungerboeck Code

output bits select the index of the desired coset Λ_0, Λ_1, Λ_2, or Λ_3. The minimum distance between the points in any of these cosets is $d_{\text{min}}(\Lambda') = 2d_{\text{min}}(\Lambda) = 4\sqrt{2}$. This distance is also the distance between the parallel transitions that are tacit in Figure B.6. Figure B.6 inherently illustrates that any two (non-parallel) paths that start and terminate in the same pair of states, must have a distance that is $d' = \sqrt{16+8+16}$, which is always greater than $4\sqrt{2}$, so that the parallel transition distance is the minimum distance for this code. This parallel-transition distance (normalized to square-root energy and ignoring constellation shape effects) is $\sqrt{2}$ better than the distance corresponding to no extra bit or just transmitting (uncoded) 4 QAM. More precisely, the coding gain is

$$\gamma = \frac{\left(d^2_{\text{min}}/\mathcal{E}_{\boldsymbol{x}_p}\right)_{\text{coded}}}{\left(d^2_{\text{min}}/\mathcal{E}_{\boldsymbol{x}_p}\right)_{\text{uncoded}}} \tag{B.108}$$

This expression evaluates to (for comparison against 4 QAM, which is also Z^N and exact for γ_f and γ_s calculations)

$$\gamma = \frac{\frac{16\cdot 2}{10}}{\frac{1}{1/2}} = 1.6 = 2 \text{ dB} \quad . \tag{B.109}$$

The fundamental coding gain is (realizing that $\bar{r}_C = \bar{r}_\Lambda + \bar{r}_G = 1.5 + .5 = 2$)

$$\gamma_f = \left(\frac{d^2_{\text{min}}}{2^{2\bar{r}_C}}\right) = \frac{32}{2^{2\cdot 2}} = 2 \text{ (3 dB)} \quad . \tag{B.110}$$

Thus, the shaping gain is then $\gamma_s = \gamma - \gamma_f = 2 - 3 = -1$ dB, which verifies according to

$$\gamma_s = \frac{2^{2\cdot 2}}{12\cdot 5}\left(2^2 - 1\right) = \frac{4}{5} = \text{ -1 dB} \ . \tag{B.111}$$

This coset code of Figure B.7 can be extended easily to the case where $b = 3$, by using the constellation in Figure B.8. In this case, $\bar{r}_C = .5 = \bar{r}_\Lambda + \bar{r}_G = 0 + .5$, $\bar{b} = 1.5$, and $\bar{\mathcal{E}} = 1.25$. This constellation contains 16 points from the scaled and translated Z^2 Lattice. The sublattice and its 4 cosets are illustrated by the labels 0,1,2, and 3 in Figure B.8. This coset code uses a circuit almost identical to that in Figure B.7, except that two bits enter the Signal Select to specify which of the 4 points in the selected coset will be the modulated signal.

The minimum distance of the coded signal is still twice the distance between points in Λ. The fundamental coding gain is

$$\gamma_f = \frac{4}{2^{2\cdot .5}} = 2 \text{ (3 dB)}. \tag{B.112}$$

The fundamental gain γ_f will remain constant at 3 dB if b further increases in the same manner; however the shaping gain γ_s will vary slightly. The shaping gain in this case is

$$\gamma_s = \frac{2^{2\cdot .5}}{12\cdot (5/4)}\left(2^3 - 1\right) = \frac{14}{15} = \text{ (-.3 dB)}. \tag{B.113}$$

The coding gain of this code with respect to 8 CR is (16/5)/(8/5)= 3dB. This code class is known as Ungerboeck's 4-state Rate 1/2 Trellis Code, and is discussed further in Section B.4.

B.3.2 Mapping By Set Partitioning

The example of the last section suggests that the basic partitioning of the signal constellation $\Lambda|\Lambda'$ can be extended to larger values of b. The general determination of Λ', given Λ, is called **mapping-by-set-partitioning.** Mapping-by-set-partitioning is instrumental to the development and understanding of coset codes.

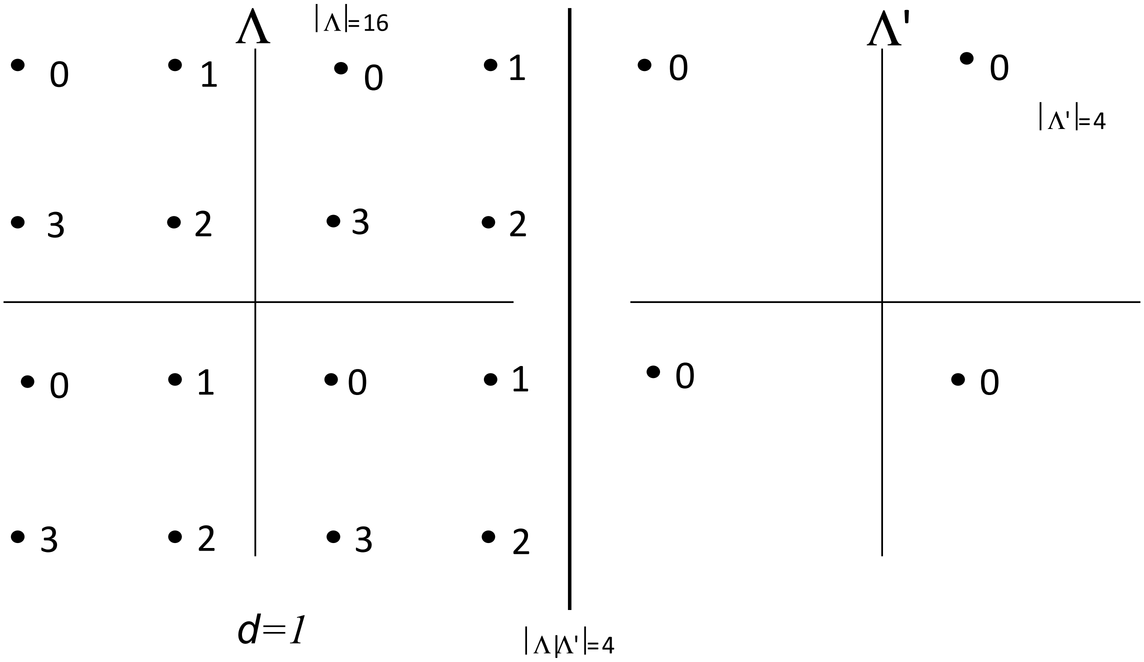


Figure B.8: Partitioning 16 QAM

B.3.2.1 Partitioning of the Integer Lattice

Example B.3.1 partitioned the 16QAM constellation used to transmit 3 bits of information per symbol. The basic idea is to divide or to partition the original constellation into two equal size parts, which both have 3 dB more distance between points than the original lattice. Figure B.9 repeats the partitioning of 16 QAM each of the partitions to generate 4 sublattices with distance 6dB better than the original Lattice.

The cosets are labeled according to an **Ungerboeck Labeling**:

> **Definition B.3.3 (Ungerboeck Labeling in two dimensions)** *An Ungerboeck labeling for a Coset Code C uses the LSB, v_0, of the encoder G output to specify which of the first 2 partitions (B0, $v_0 = 0$ or B1 $v_0 = 1$) contains the selected coset of the sublattice Λ', and then uses v_1 to specify which of the next level partitions (C0,C2,C1,C3) contains the selected coset of the sublattice, and finally when necessary, v_2 is used to select which of the 3rd level partitions is the selected coset of the sublattice. This last level contains 8 cosets, D0,D4,D2,D6,D1,D5,D3, and D7.*

The remaining bits, v_{k+r}, ..., v_{b+r-1} are used to select the point within the selected coset. The partitioning process can be continued for larger constellations, but is not of practical use for two-dimensional codes.

In practice, mapping by set partitioning is often used for $N = 1$, $N = 2$, $N = 4$, and $N = 8$. One-dimensional partitioning halves a PAM constellation into sets of “every other point,” realizing a 6dB increase in intra-partition distance for each such halving. In 4 and 8 dimensions, which will be considered later, the distance increase is (on the average) 1.5 dB per partition and .75 dB per partition, respectively.

B.3.2.2 Partition Trees and Towers

Forney’s **Partition trees** and **Towers** are alternative more mathematical descriptions of mapping by set partitioning and the Ungerboeck Labeling process.

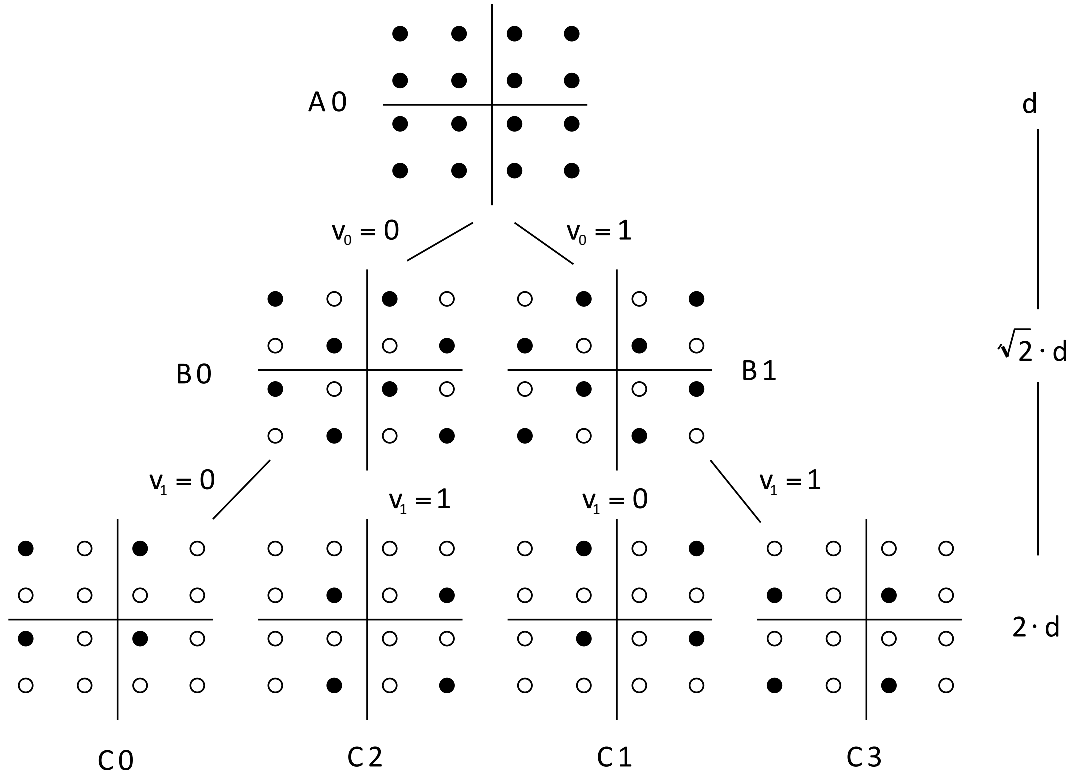


Figure B.9: Illustration of Mapping by Set Partitioning for 16QAM Constellation

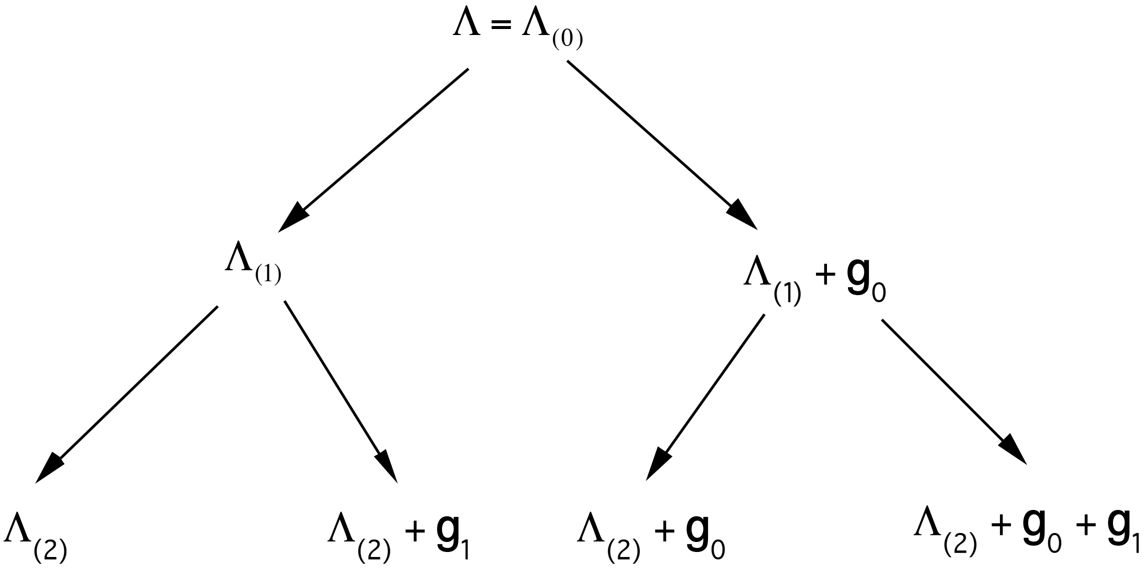


Figure B.10: Partition tree.

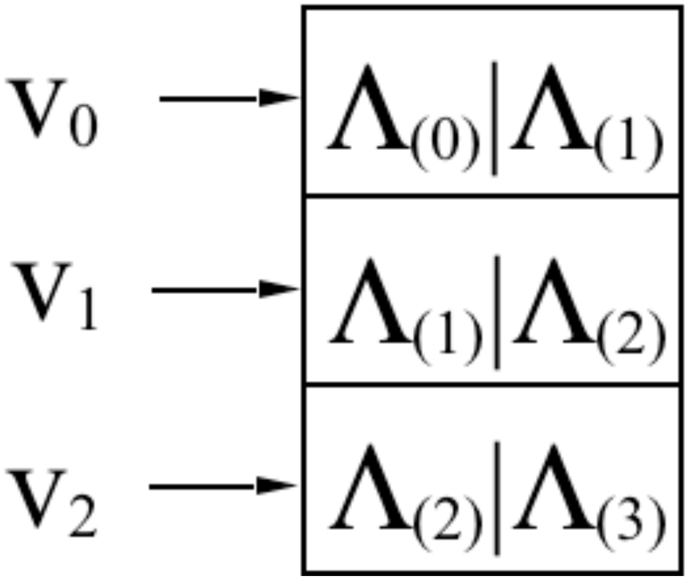

V0
V1
V2
$\Lambda_{(0)}|\Lambda_{(1)}$
$\Lambda_{(1)}|\Lambda_{(2)}$
$\Lambda_{(2)}|\Lambda_{(3)}$

Figure B.11: Partition tower.

The partition tree is shown in Figure B.10. Each bit of the convolutional encoder (G) output is used to delineate one of two cosets of the parent lattice at each stage in the tree. The constant vector that is added to one of the partitions to get the other is called a **coset leader**, and is mathematically denoted as $\boldsymbol{g}_i$. The specification of a vector point that represents any coset in the i^{th} stage of the tree is

$$\boldsymbol{x} = \Lambda' + \sum_{i=0}^{k+r_G-1} v_i \boldsymbol{g}_i = \Lambda' + \boldsymbol{v} \begin{bmatrix} \boldsymbol{g}_{k+r_G-1} \\ \vdots \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_0 \end{bmatrix} = \Lambda' + \boldsymbol{vG} \quad , \tag{B.114}$$

where $\boldsymbol{G}$ is a "generator" for the coset code. Usually in practice, a constant offset $\boldsymbol{a}$ is added to all $\boldsymbol{x}$ so that 0 (a member of all lattices) is not in the constellation, and the constellation has zero mean. The final $b - k$ bits will specify an offset from $\boldsymbol{x}$ that will generate our final modulation symbol vector. The set of all possible binary combinations of the vectors $\boldsymbol{g}_i$ is often denoted

$$[\Lambda|\Lambda'] \stackrel{\Delta}{=} \{\boldsymbol{vG}\} \tag{B.115}$$

Thus,

$$\Lambda = \Lambda' + [\Lambda|\Lambda'] \quad , \tag{B.116}$$

symbolically, to abbreviate that every point in Λ can be (uniquely) decomposed as the sum of a point in Λ' and one of the "coset leaders" in the set $[\Lambda|\Lambda']$, or recursively

$$\Lambda_{(i)} = \Lambda_{(i+1)} + \left[\Lambda_{(i)}|\Lambda_{(i+1)}\right] \quad . \tag{B.117}$$

There is thus a **chain rule**

$$\Lambda_{(0)} = \Lambda_{(2)} + \left[\Lambda_{(1)}|\Lambda_{(2)}\right] + \left[\Lambda_{(0)}|\Lambda_{(1)}\right] \tag{B.118}$$

or

$$\left[\Lambda_{(0)}|\Lambda_{(2)}\right] = \left[\Lambda_{(0)}|\Lambda_{(1)}\right] + \left[\Lambda_{(1)}|\Lambda_{(2)}\right] \tag{B.119}$$

where $\left[\Lambda_{(i)}|\Lambda_{(i+1)}\right] = \{0,\ \boldsymbol{g}_i\}$ and $\left[\Lambda_{(0)}|\Lambda_{(2)}\right] = \{0,\ \boldsymbol{g}_0,\ \boldsymbol{g}_1,\ \boldsymbol{g}_0 + \boldsymbol{g}_1\}$. This concept is probably most compactly described by the partition tower of Forney, which is illustrated in Figure B.11.

The partition chain is also compactly denoted by

$$\Lambda_{(0)} \left|\Lambda_{(1)}\right| \Lambda_{(2)} \left|\Lambda_{(3)}\right| .... \tag{B.120}$$

EXAMPLE B.3.2 (Two-dimensional integer lattice partitioning) The initial lattice $\Lambda_{(0)}$ is Z^2. Partitioning selects "every other" point from this lattice to form the lattice D_2, which has only those points in Z^2 that have even squared norms (see Figures B.9 and B.5). Mathematically, this partitioning can be written by using the rotation matrix

$$R_2 \triangleq \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{B.121}$$

as

$$D_2 = R_2 Z^2 \quad , \tag{B.122}$$

where the multiplication in (B.122) symbolically denotes taking each point in Z^2 and multiplying by the rotation matrix R_2 to create a new set of points that will also be a lattice. D_2 is also a sublattice of Z^2 and therefore partitions Z^2 into two subsets. Thus in partitioning notation:

$$Z^2 \left| D_2 \right. \quad . \tag{B.123}$$

The row vector that can be added to D_2 to get the other coset is $\boldsymbol{g}_0 = [0\ 1]$. So, $Z^2 \left| D_2 \right.$ with coset leaders $\left[Z^2 \left| D_2 \right. \right] = \{[0\ 0],\ [0\ 1]\}$. D_2 decomposes into two sublattices by again multiplying by the rotation operator R_2

$$2Z^2 = R_2 D_2 \quad . \tag{B.124}$$

Points in $2Z^2$ have squared norms that are multiples of 4. Then, $D_2 \left| 2Z^2 \right.$ with $\left[D_2 \left| 2Z^2 \right. \right] = \{[0\ 0],\ [1\ 1]\}$. Thus, $\left[Z^2 \left| 2Z^2 \right. \right] = \{[0\ 0],\ [0\ 1],\ [1\ 0]\ [1\ 1]\}$. The generator for a coset code with convolutional codewords $[v_1(D)\ v_0(D)]$ as input is then

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad . \tag{B.125}$$

Partitioning continues by successive multiplication by R_2: $Z^2 \left| D_2 \right| 2Z^2 \left| 2D_2 \right| 4Z^2 \ldots.$

B.4 One- and Two-dimensional Trellis Codes

Ungerboeck's two most famous codes are in wide use and used for illustrative purposes in this section.

B.4.1 Rate 1/2 Code

The earlier 4-state rate 1/2 code discussed in Section B.3 can be generalized as shown in Figure B.12. The encoder matrix $G = [1 + D^2 \; D]$ can be equivalently written as a systematic encoder

$$G_{sys} = \left[1 \; \frac{D}{1+D^2} \right] \quad , \tag{B.126}$$

which is shown in Figure B.12. The parity matrix H appears in Figure B.12 because most trellis codes are more compactly expressed in terms of H. In this code $r_G = 1$ and 2^{b+1} points are taken from the Lattice coset $Z^2 + \left[\frac{1}{2} \; \frac{1}{2}\right]$ to form the constellation Λ, while the sublattice that is used to partition Λ is $\Lambda' = 2Z^2$, upon which there are 2^{b-1} constellation points, and there are 4 cosets of Λ' in Λ, $|\Lambda/\Lambda'| = 4$. The fundamental gain remains at

$$\gamma_f = \left(\frac{d^2_{\text{min}}(2Z^2)}{\mathcal{V}(Z^2) \cdot 2} \right) / \left(\frac{d^2_{\text{min}}(Z^2)}{\mathcal{V}(Z^2)} \right) = \frac{4}{2} / \frac{1}{1} = 2 = 3 \text{ dB} \quad . \tag{B.127}$$

Shaping gain is again a function of the constellation shape, and is not considered to be part of the fundamental gain of any code. For any number of input bits, b, the structure of the code remains mainly the same, with only the number of points within the cosets of $\Lambda' = 2Z^2$ increasing with 2^b.

The partition chain for this particular code is often written $Z^2 | D_2 | 2Z^2$.

B.4.2 A simple rate 2/3 Trellis Code

In the rate 2/3 code of interest, the encoder G will be a rate 2/3 encoder. The objective is to increase fundamental gain beyond 3dB, which was the parallel transition distance in the rate 1/2 code of Section B.3. Higher gain necessitates constellation partitioning by one additional level/step to ensure that the parallel transition distance will now be 6dB, as in Figure B.13. Then, the minimum distance will usually occur between two longer length sequences through the trellis, instead of between parallel transitions. The mapping-by-set-partitioning principle of the Section B.3.2 extends one more level to the chain $Z^2 | D_2 | 2Z^2 | 2D_2$, which is illustrated in detail in Figure B.13 for a 16 SQ QAM constellation. Figure B.14 shows a trellis for the successive coset selection from D-level sets in Figure **??** and also illustrates an example of the worst-case path for computation of d_{min}.

The worst case path has distance

$$d_{\text{min}} = \sqrt{2+1+2} = \sqrt{5} < \sqrt{8} \quad . \tag{B.128}$$

The fundamental gain is thus

$$\gamma_f = \frac{d^2_{\text{min}}}{2^{2\bar{r}_C}} = \frac{5}{2^{2(1/2)}} = 2.5 = 4 \text{ dB} \quad . \tag{B.129}$$

Essentially, this rate 2/3 code with 8 states, has an extra 1 dB of coding gain. It is possible to yet further increase the gain to 6dB by using a trellis with more states, as Section B.4.3 will show.

Recognizing that this trellis is the same that was analyzed in Sections **??** and **??** or equivalently reading the generator from the circuit diagram in Figure B.15,

$$G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{B.130}$$

Since this is a systematic realization,

$$G_{sys} H' = 0 = \left[I \;\; h^T \right] \begin{bmatrix} h^T \\ I \end{bmatrix} \quad , \tag{B.131}$$

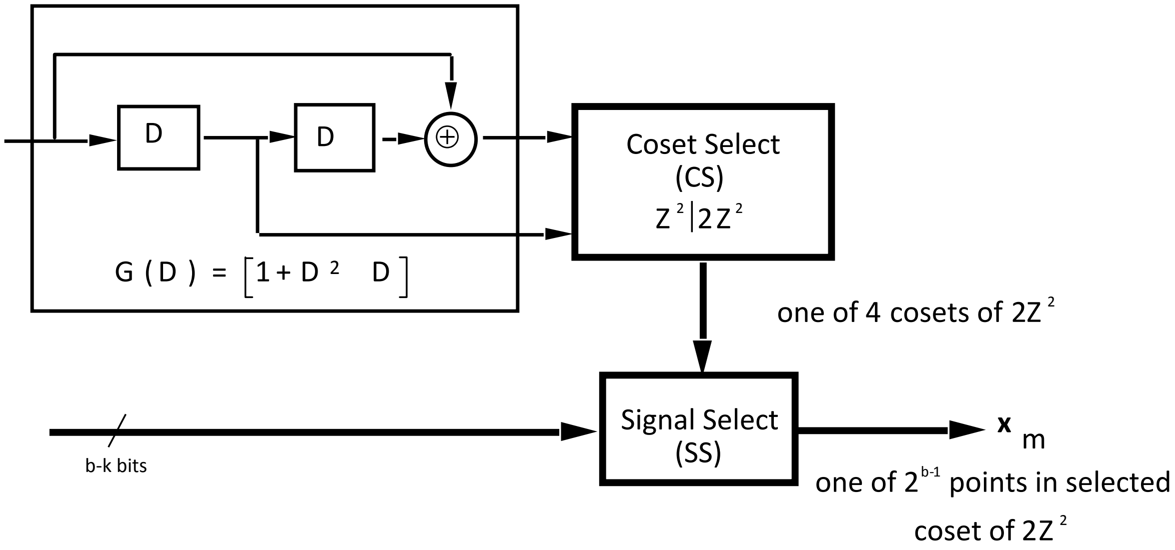


Figure B.12: Ungerboeck's 4-state rate 1/2 Trellis Code

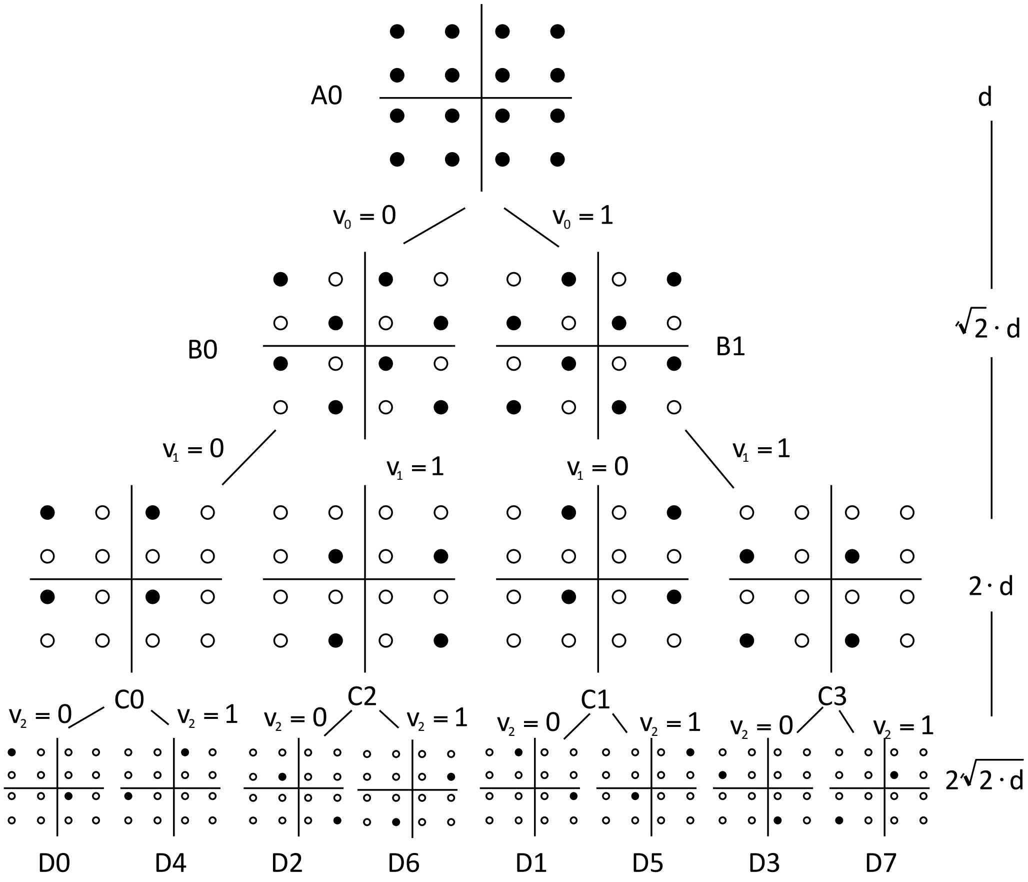


Figure B.13: D-level partitioning of 16 QAM.

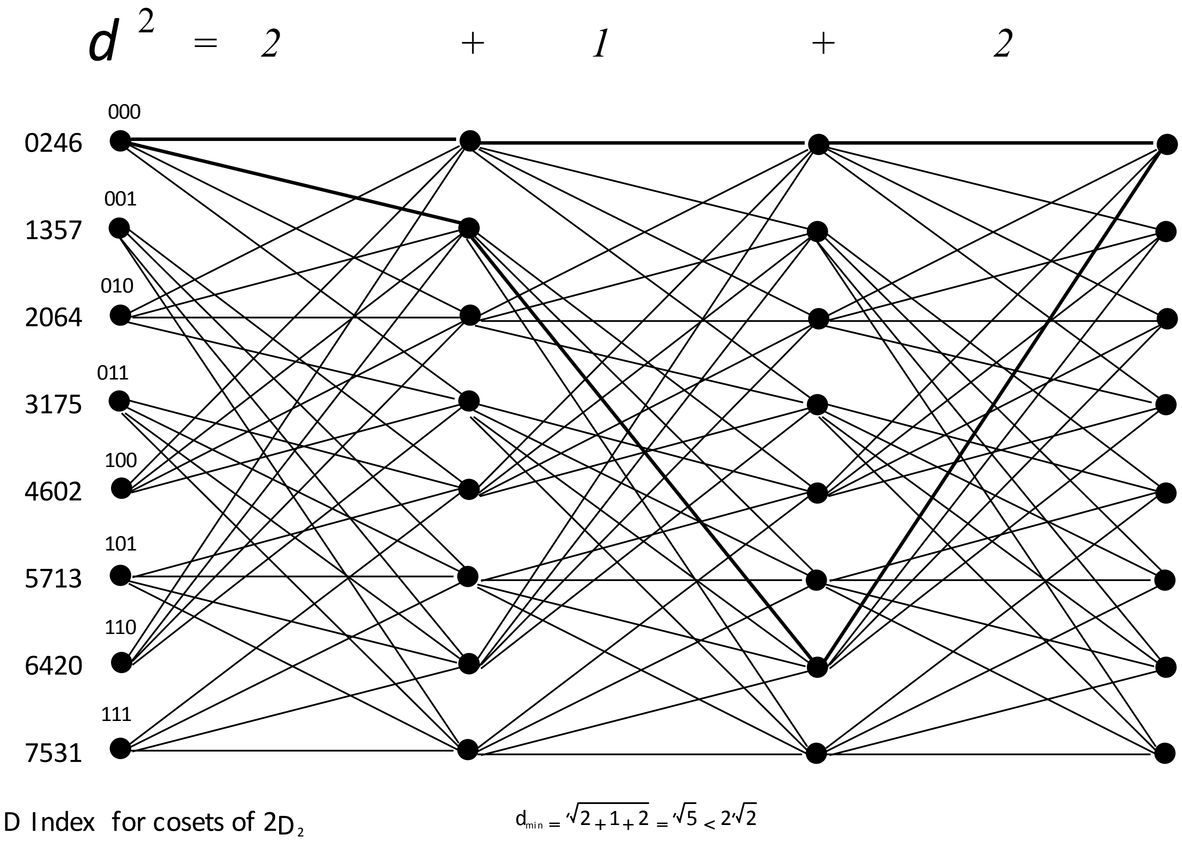


Figure B.14: Trellis for 8-state rate 2/3 Trellis Code ($\gamma_f = 3$ dB)

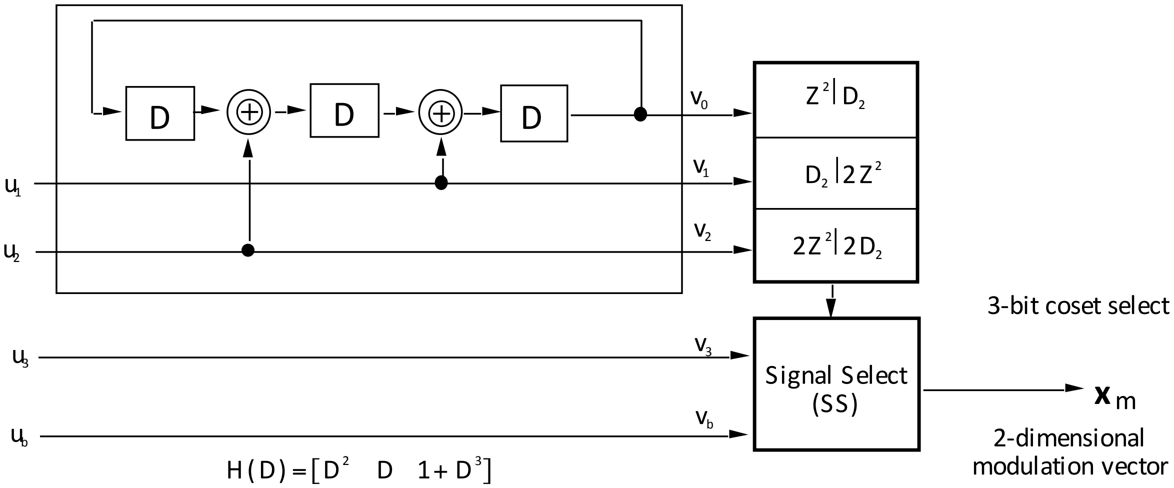


Figure B.15: Encoder for 8-state rate 2/3 trellis code ($\gamma_f = 4$ dB).

so that the last $n-k=1$ column(s) are the rows of the parity matrix, or

$$H(D)=[h\ \ 1] \quad . \tag{B.132}$$

In the case of the code in question (after clearing the denominator),

$$H(D)=\begin{bmatrix} D^2 & D & 1+D^3 \end{bmatrix} \quad . \tag{B.133}$$

With trellis codes of rate $k/k+1$, the extra bit is named v_0 instead of v_{k+1}. A variant of the code in Figure B.15 is used in the voiceband modem standards v.32 and v.32bis.

B.4.3 Code Design in One and Two Dimensions

Previous sections specifically studied two 2-dimensional trellis codes in detail: the first being a rate 1/2 4-state code with fundamental gain $\gamma_f=3$ dB, and the second being a rate 2/3 8-state code with $\gamma_f=4$ dB. There are many yet more powerful (higher γ_f) codes that have been mainly tabulated by Ungerboeck and Forney[2] that this subsection lists again shortly. There are also codes designed for the one-dimensional channel, which appear next.

B.4.3.1 One-Dimensional Trellis Codes

For one dimensional codes to have up to 6 dB of fundamental gain, the designer need only partition twice, so that $\Lambda'=\Lambda_{(2)}$ to realize a minimum separation between any two parallel transitions that is 6 dB higher than uncoded one-dimensional PAM. Stated more precisely, the partition chain for the one-dimensional trellis codes is, with $\Lambda=Z$, $Z|2Z|4Z$. The corresponding minimum distances between points are $d_{\min}(Z)=1$, $d_{\min}(2Z)=2$, and $d_{\min}(4Z)=4$, respectively, and $r_G=1$ for the one-dimensional codes in this chapter. Since $r_G=1$ implies a doubling of the constellation size $|\Lambda|$, with respect to uncoded transmission, the maximum fundamental gain will be limited to

$$\gamma_f \le 16/(2^{2\cdot 1})=6\text{ dB} \quad , \tag{B.134}$$

because the parallel transition separation is never more than $d^2=16$. Since $G(D)$ must then be a rate 1/2 code, then $G(D)$ and $H(D)$ are both 1×2 matrices. In one-dimensional codes, the actual signal set Λ is offset to eliminate any nonzero mean, and has equal numbers of positive and negative points, so that $\Lambda \to Z+\frac{1}{2}$. $d^2_{\min}$ for any such code must be an integer because it is a sum of squared integers.

Table B.1 lists most of the best-known one-dimensional codes (and was essentially copied from Forney's 1988 Coset Codes I paper). Recall that $\bar{N}_e=N_e$ is the normalized number of nearest neighbors. The quantities $\bar{N}_1$ and $\bar{N}_2$ are the numbers of next-to-nearest neighbors (with squared distance $d^2=d^2_{\min}+1$), and next-to-next-to-nearest neighbors (with squared distance $d^2_{\min}+2$). The effective gain $\tilde{\gamma}_f$ is the fundamental gain of the code, reduced by .2dB for each factor of 2 increase in nearest neighbor over the minimum of 2 nearest neighbors per dimension, which occurs for uncoded PAM transmission. $\bar{N}_D$ is a measure of decoder complexity that will be described in Section B.4.4. An example of the use of these tables appears in Subsection B.4.3.4.

For more complete understanding of the entries in the one-dimensional coding tables, Table B.2 summarizes some partitioning results in one-dimension in Table B.2. The subscript on a partition refers to the dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck labeling. The distances in the table correspond to

$$A_1^0 = Z \tag{B.135}$$
$$B_1^0 = 2Z \tag{B.136}$$
$$B_1^1 = 2Z+1 \tag{B.137}$$
$$C_1^0 = 4Z \tag{B.138}$$
$$C_1^1 = 4Z+1 \tag{B.139}$$
$$C_1^2 = 4Z+2 \tag{B.140}$$
$$C_1^3 = 4Z+3 \quad . \tag{B.141}$$

[2] The author would like to acknowledge help from Nyles Heise of IBM Almaden Research who updated and corrected some of these tables - Heise's corrections on the tables of Unberboeck and Forney are included here.

2^ν	h_1	h_0	$d^2_{\min}$	γ_f	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\bar{N}_3$	$\bar{N}_4$	$\tilde{\gamma}_f$	$\bar{N}_D$
4	2	5	9	2.25	3.52	4	8	16	32	64	3.32	12
8	04	13	10	2.50	3.98	4	8	16	40	72	3.78	24
16	04	23	11	2.75	4.39	8	8	16	48	80	3.99	48
16	10	23	11	2.75	4.39	4	8	24	48	80	4.19	48
32	10	45	13	3.25	5.12	12	28	56	126	236	4.60	96
64	024	103	14	3.50	5.44	36	0	90	0	420	4.61	192
64	054	161	14	3.50	5.44	8	$\underline{32}$	66	84	236	4.94	192
128	126	235	16	4.00	6.02	66	0	256	0	1060	5.01	384
128	160	267	15	3.75	5.74	8	34	$\underline{100}$	164	344	5.16	384
128	124	207	14	3.50	5.44	4	8	14	56	136	5.24	384
256	362	515	16	4.00	6.02	2	32	$\underline{80}$	132	268	5.47	768
256	370	515	15	3.75	5.74	4	6	$\underline{40}$	68	140	5.42	768
512	0342	1017	16	4.00	6.02	2	0	56	0	$\underline{332}$	5.51	1536

Table B.1: One-Dimensional Trellis Codes and Parameters
(Underlined quantities correspond to cases where worst-case performance is caused by large next-to-nearest neighbor counts.)

	A_1^0			
$d_{\min}$ w.r.t. A_1^0	1			
N_e w.r.t. A_1^0	2			
	B_1^0		B_1^1	
$d_{\min}$ w.r.t. B_1^0	2		1	
N_e w.r.t. B_1^0	2		2	
	C_1^0	C_1^2	C_1^1	C_1^3
$d_{\min}$ w.r.t. C_1^0	4	2	1	1
N_e w.r.t. C_1^0	2	2	1	1

Table B.2: One-Dimensional Partitioning

2^ν	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\bar{N}_3$	$\bar{N}_4$	$\tilde{\gamma}_f$	$\bar{N}_D$
4	-	2	5	4	2	3.01	2	16	64	256	1024	3.01	8
8	04	02	11	5	2.5	3.98	8	36	160	714	3144	3.58	32
16	16	04	23	6	3	4.77	28	80	410	1952	8616	4.01	60
32	10	06	41	6	3	4.77	8	52	202	984	4712	4.37	116
32	34	16	45	6	3	4.77	4	$\underline{64}$	202	800	4848	4.44	116
64	064	016	101	7	3.5	5.44	28	130	504	2484	12236	4.68	228
64	060	004	143	7	3.5	5.44	24	146	592	2480	12264	4.72	228
64	036	052	115	7	3.5	5.44	20	126	496	2204	10756	4.78	228
128	042	014	203	8	4	6.02	172	0	2950	0	73492	4.74	451
128	056	150	223	8	4	6.02	86	312	1284	6028	29320	4.94	451
128	024	100	245	7	3.5	5.44	4	$\underline{94}$	484	1684	8200	4.91	451
128	164	142	263	7	3.5	5.44	4	$\underline{66}$	376	1292	6624	5.01	451
256	304	056	401	8	4	6.02	22	152	658	2816	$\underline{13926}$	5.23	900
256	370	272	417	8	4	6.02	18	154	612	2736	$\underline{13182}$	5.24	900
256	274	162	401	7	3.5	5.44	2	$\underline{32}$	124	522	2732	5.22	900
512	0510	0346	1001	8	4	6.02	2	64	350	1530	$\underline{6768}$	5.33	1796

Table B.3: Two-Dimensional Trellis Codes and Parameters

(Underlined quantities correspond to cases where worst-case performance is caused by large next-to-nearest neighbor counts).

B.4.3.2 Two-Dimensional Codes

Two-dimensional codes use 3-level partitioning[3], so that $\Lambda' = \Lambda_{(3)}$ to realize a minimum separation between any two parallel transitions that is 6dB higher than uncoded two-dimensional QAM. Stated more precisely, the partition chain for the two-dimensional trellis codes of interest is, with $\Lambda = Z^2$, $Z^2|D_2|2Z^2|2D_2$. The corresponding minimum distances between points are $d_{\text{min}}(Z^2) = 1$, $d_{\text{min}}(D_2) = \sqrt{2}$, $d_{\text{min}}(2Z^2) = 2$, and $d_{\text{min}}(2D_2) = 2\sqrt{2}$ respectively, and $r_G = 1$ for the two-dimensional codes presented here. Since $r_G = 1$ implies a doubling of the two-dimensional constellation size $|\Lambda|$, with respect to uncoded transmission, the maximum fundamental gain will be limited to

$$\gamma_f \le 8/2 = 6\text{dB} \quad . \tag{B.142}$$

Since $G(D)$ must then be a rate 2/3 code, then $G(D)$ is a 2×3 matrix and $H(D)$ is a 1×3 matrix, making $H(D)$ the more compact description. In the two-dimensional codes, the actual signal set Λ is offset to eliminate any nonzero mean, and has equal numbers of points in each quadrant, so that $\Lambda \to Z^2 + [\frac{1}{2}, \frac{1}{2}]$. d^2_{min} for any code must be an integer because it is a sum of integers.

Table B.3 lists most of the best-known two-dimensional codes (and was also essentially copied from Forney's Coset Codes I paper). Recall that $\bar{N}_e$ is the normalized number of nearest neighbors. The quantities $\bar{N}_1$ and $\bar{N}_2$ mean the same thing they did for the one-dimensional codes, allowing comparisons on a per-dimensional basis between one and two dimensional codes.

For more complete understanding of the entries in the two-dimensional coding tables, Table B.4 summarizes some partitioning results in two-dimensions. The subscript on a partition refers to the dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck

[3] With the only exception being the 4-state 3dB code that was already studied

	A_2^0			
$d_{\min}$ w.r.t. A_2^0	1			
N_e w.r.t. A_2^0	4			
	B_2^0		B_2^1	
$d_{\min}$ w.r.t. B_2^0	$\sqrt{2}$		1	
N_e w.r.t. B_2^0	4		4	
	C_2^0	C_2^2	C_2^1	C_2^3
$d_{\min}$ w.r.t. C_2^0	2	$\sqrt{2}$	1	1
N_e w.r.t. C_2^0	4	4	2	2
	D_2^0	D_2^2	D_2^1	D_2^3
$d_{\min}$ w.r.t. D_2^0	$\sqrt{8}$	$\sqrt{2}$	1	1
N_e w.r.t. D_2^0	4	2	1	1
	D_2^4	D_2^6	D_2^5	D_2^7
$d_{\min}$ w.r.t. D_2^0	2	$\sqrt{2}$	1	1
N_e w.r.t. D_2^0	4	2	1	1

Table B.4: Two-Dimensional Partitioning

labeling. The distances in the table correspond to

$$
\begin{aligned}
A_2^0 &= Z^2 && \text{(B.143)} \\
B_2^0 &= RZ^2 && \text{(B.144)} \\
B_2^1 &= RZ^2 + [1,0] && \text{(B.145)} \\
C_2^0 &= 2Z^2 && \text{(B.146)} \\
C_2^1 &= 2Z^2 + [1,0] && \text{(B.147)} \\
C_2^2 &= 2Z^2 + [1,1] && \text{(B.148)} \\
C_2^3 &= 2Z^2 + [0,1] = 2Z^2 + [2,1] && \text{(B.149)} \\
D_2^0 &= 2RZ^2 && \text{(B.150)} \\
D_2^1 &= 2RZ^2 + [1,0] && \text{(B.151)} \\
D_2^2 &= 2RZ^2 + [1,-1] && \text{(B.152)} \\
D_2^3 &= 2RZ^2 + [2,-1] && \text{(B.153)} \\
D_2^4 &= 2RZ^2 + [0,-2] && \text{(B.154)} \\
D_2^5 &= 2RZ^2 + [1,-2] && \text{(B.155)} \\
D_2^6 &= 2RZ^2 + [1,-3] && \text{(B.156)} \\
D_2^7 &= 2RZ^2 + [0,1] = 2RZ^2 + [2,-3] \quad . && \text{(B.157)}
\end{aligned}
$$

B.4.3.3 Phase-Shift Keying Codes

Although, PSK codes fall properly outside the domain of coset codes as considered here - gains can be computed and codes found for two cases of most practical interest. Namely 4PSK/8PSK systems and 8PSK/16PSK systems. The corresponding code tables are illustrated in Tables B.5 and B.6. Partitioning proceeds as in one-dimension, except that dimension is wrapped around a circle of circumference 2^{b+1}.

All PSK trellis codes have their gain specified with respect to the uncoded circular constellation - that is with respect to QPSK for $b = 2$ or to 8PSK for $b = 3$.

2^ν	h_2	h_1	h_0	γ	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\tilde{\gamma}$
4	-	2	5	2	3.01	.5	?	?	3.41
8	04	02	11	2.291	3.60	2	?	?	3.80
16	16	04	23	2.588	4.13	1.15	?	?	4.29
32	34	16	45	2.877	4.59	2	?	?	4.59
64	066	030	103	3.170	5.01	2.5	?	?	4.95
128	122	054	277	3.289	5.17	.25	?	?	5.67?
256	130	072	435	3.758	5.75	.75	?	?	6.03?

Table B.5: 4PSK/8PSK Trellis Codes and Parameters
(Effective gains are suspect, as next-to-nearest neighbor counts are not presently available.)

2^ν	h_2	h_1	h_0	γ	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\tilde{\gamma}$
4	–	2	5	2.259	3.54	2	?	?	3.54
8	–	04	13	2.518	4.01	2	?	?	4.01
16	–	04	23	2.780	4.44	4	?	?	4.24
32	–	10	45	3.258	5.13	4	?	?	4.93
64	–	024	103	3.412	5.33	1	?	?	5.53?
128	–	024	203	3.412	5.33	1	?	?	5.53?
256	374	176	427	3.556	5.51	4	?	?	5.31?

Table B.6: 8PSK/16PSK Trellis Codes and Parameters
Effective gains are suspect, as next-to-nearest neighbor counts are not presently available.

B.4.3.4 Design Examples

This subsection presents two design examples to illustrate the use of Tables B.1 and B.3.

EXAMPLE B.4.1 (32CR Improved by 4.5dB) A data transmission system transmits 5 bits/2D-symbol over an AWGN channel with channel SNR=19.5dB. This SNR is only sufficient, using 32CR QAM, to achieve a probability of error

$$P_e = 4\left(1 - \frac{1}{\sqrt{2\cdot 32}}\right) Q\left[\sqrt{\frac{3\mathrm{SNR}}{(31/32)32 - 1}}\right] \approx 4Q(2.985) \approx .0016 \quad , \tag{B.158}$$

which is (usually) insufficient for reliable data transmission. An error rate of approximately 10^{-6} is desired. To get this improved error rate, the applied code needs to increase the SNR in (B.158) by approximately 4.5dB. Before using the tables, the designer computes the shaping gain (or loss) from doubling the signal set size from 32CR to 64SQ (presuming no more clever signal constellation with 64 points is desirable for this example) as

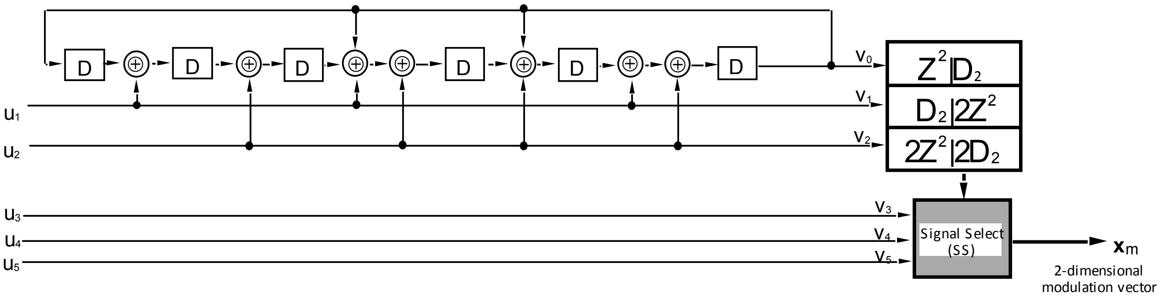
$$\gamma_s(32CR) = 10\log_{10}\left(\frac{1\cdot(2^5-1)}{12\cdot(5/2)}\right) = .14dB \tag{B.159}$$

and

$$\gamma_s(64QAM) = 10\log_{10}\left(\frac{2\cdot(2^5-1)}{12\cdot(10.5/2)}\right) = -0.07dB \quad , \tag{B.160}$$

thus the design loses .21dB in going to the 64SQ QAM constellation for trellis coding with respect to the 32CR QAM. This is because 32CR QAM is closer to a circular boundary than is 64 SQ QAM.

Table B.3 contains two 64-state codes that achieve an effective coding gain of at least 4.72dB. Since the 2 satisfactory codes listed have the same complexity, a better choice is the last, which has effective gain 4.78dB. Taking the shaping gain penalty gain, the full gain of this



$$H = \left[\; D^4 + D^3 + D^2 + D \quad D^5 + D^3 + D \quad D^6 + D^3 + D^2 + 1 \;\right]$$

Figure B.16: Circuit for 64QAM code with 4.57dB gain.

code with respect to 32CR is 4.78-.21 = 4.57 dB, and the error probability is then

$$P_e \approx 4Q\left[\sqrt{\frac{3(\text{SNR}+4.57\text{dB})}{(31/32)32-1}}\right] \approx 4Q(5.05) \approx 9\times 10^{-7} \quad , \tag{B.161}$$

which is below the desired 10^{-6}. From the table, $h_2 = 036$, $h_1 = 052$, and $h_0 = 115$, so that

$$H = \left[D^4 + D^3 + D^2 + D \quad D^5 + D^3 + D \quad D^6 + D^3 + D^2 + 1\right] \tag{B.162}$$

or equivalently in systematic form

$$H = \left[\frac{D^4 + D^3 + D^2 + D}{D^6 + D^3 + D^2 + 1} \quad \frac{D^5 + D^3 + D}{D^6 + D^3 + D^2 + 1} \quad 1\right] \quad . \tag{B.163}$$

(Recall that $GH^* = 0$, so that $G = [I\ h^*]$ when $H = [h\ 1]$.) The circuit for implementation of this code, in systematic form, is illustrated in Figure B.16 and a labeled (using Ungerboeck labeling) 64QAM constellation for the code is shown in Figure B.17.

The second example uses a 1-dimensional code in conjunction with the 1+.9D channel that was studied extensively in EE379A. For this system, given developments in this text to this point, the best means of applying trellis coding is to use a decision feedback equalizer (infinite length) to eliminate ISI, and for which our mean-square error sequence will be white (but not necessarily Gaussian, which this text's analysis will assume anyway).

EXAMPLE B.4.2 ($1 + .9D^{-1}$ Revisited for Code Concatenation) From previous study of this example in earlier chapters, the SNR for the MMSE-DFE was 8.4dB on this $1+.9D^{-1}$ channel with $\text{SNR}_{mfb} = 10$dB and 1 bit/T transmission. The corresponding error rate on such a system would be completely unacceptable in most applications, so improvement is desirable. A one-dimensional 256-state trellis code from Table B.1 has effective gain 5.47dB. The shaping gain loss in going from 1bit/T to the 4 levels/T necessary in this code is easily computed as 4/5=-.97dB. The coding gain for this application would then be 5.47-.97=4.50dB. The probability of error for the coded system would then be

$$P_e = 2\cdot Q(8.4 + 4.5dB) = 2\cdot Q(4.41) = 10^{-5} \quad . \tag{B.164}$$

Unfortunately, error propagation in the internal DFE is now more likely, not only because an uncoded system would have had a high probability of error of about .01, but also because the enlarged constellation from which a DFE would have to make instantaneous decisions now has more points and smaller symbol-by-symbol-detection distance. A Laroia precoder here with the enlarged constellation would only lose a few tenths of a dB. The parity matrix is 1×2 and can be read from Table B.1 as

$$H = \left[D^7 + D^6 + D^5 + D^4 + D \quad D^8 + D^6 + D^3 + D^2 + 1\right] \tag{B.165}$$

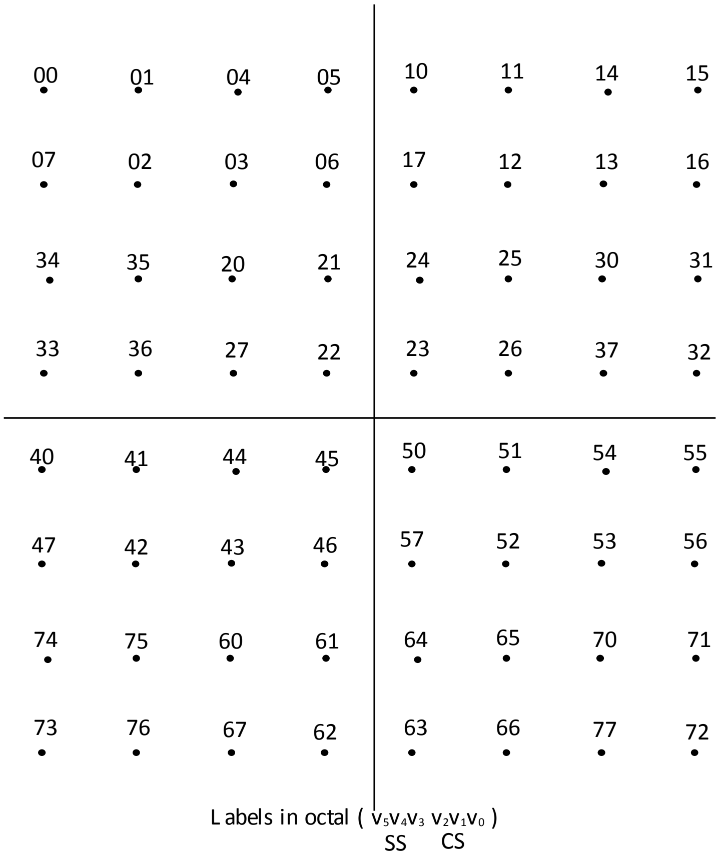


Figure B.17: Constellation for 64QAM code example with 4.57dB gain.

which corresponds to a systematic G of

$$G_{sys} = \left[1 \quad \frac{D^7 + D^6 + D^5 + D^4 + D}{D^8 + D^6 + D^3 + D^2 + 1} \right] \quad . \tag{B.166}$$

B.4.3.5 Decision-Feedback Sequence Estimation

Decision-Feedback Sequence Estimation (DFSE) essentially eliminates error propagation when DFE's are used with coset codes. In DFSE, the survivor into each state is used to determine 2^ν possible feedback-section outputs, one for each state. The ISI-subtraction associated with that correct survivor is then used in computing the branch metric into each state.

B.4.4 Decoder Complexity Measures

The implementation complexity of a trellis or lattice code depends heavily on the details of the application. Nevertheless, it is possible to associate with each coset code, a meaningful measure of complexity. This measure is mainly relative and used for comparing the use of two different codes in the same application. The measure used in this book is equivalent to one introduced by Forney in 1988.

This measure is computed for a maximum-likelihood (Viterbi for trellis codes) detector and ignores encoder complexity. The complexity measure counts the number of additions and number of comparisons that need to be performed in the straightforward implementation of the decoder and adds these two numbers together. This measure is essentially a count of the number of instruction cycles that are required to implement the decoder on a programmable signal processor.

> **Definition B.4.1 (Decoder Complexity)** *The quantity N_D is called the* **decoder complexity** *for a coset code and is defined to be the total number of additions and comparisons that are needed to implement the decoder in the straightforward maximum-likelihood implementation. The* **normalized decoder complexity** *is $\bar{N}_D \triangleq N_D/N$.*

In decoding a coset code, the N-dimensional channel output vector $\boldsymbol{y}$ is used to resolve which of the possible points in each of the cosets used by the code is closest to this received sample $\boldsymbol{y}$. This step chooses the parallel transition between states that is more likely than the other such parallel transitions. Once these parallel transitions (one for each coset) have been chosen, sequence detection (via the Viterbi algorithm) chooses the sequence of cosets that was most likely.

For one-dimensional coset codes, the resolution of the closet point in each of the 4 cosets is trivial, and is essentially a truncation of the received vector $\boldsymbol{y}$, so this operation is not included in N_D. Then, for a rate $k/(k+r_G)$ code with 2^ν states, the decoder requires 2^k adds and $2^k - 1$ binary comparisons for each state per received one-dimensional output. This is a total of

$$N_D(one-dimensional) = 2^\nu \left(2^k + 2^k - 1\right) = 2^{\nu+k}\left(2 - 2^{-k}\right) \quad . \tag{B.167}$$

This computational count is independent of b because it ignores the truncation associated with choosing among parallel transitions. This also permits the inclusion of N_D in the tables of Section B.4.3, which do not depend on b.

For two-dimensional codes, the parallel-transition resolution that was trivial in one dimension now requires one addition for each two-dimensional coset. While table look-up or truncation can be used to resolve each of the one-dimensional components of the two-dimensional $\boldsymbol{y}$ for every coset, N_D includes an operation for the addition of the component one-dimensional metrics to get the two-dimensional metric. Since there are 2^{k+r_G} cosets in the coset code, then

$$N_D(two-dimensional) = 2^\nu \left(2^k + 2^k - 1\right) + 2^{k+r_G} = 2^{\nu+k}\left(2 - 2^{-k}\right) + 2^{k+r_G} \quad , \tag{B.168}$$

or

$$\bar{N}_D = 2^{\nu+k}\left(1 - 2^{-k-1}\right) + 2^{k+r_G-1} \quad . \tag{B.169}$$

The computation of N_D for the higher dimensional codes that is discussed later in this chapter is similar, except that the resolution of the parallel transitions becomes increasingly important and complex as the code dimensionality grows.

B.5 Multidimensional Trellis Codes

Multidimensional lattices can be combined with trellis coding to get a larger coding gain for a given complexity, and can also reduce the expansion of the constellation slightly (which may be of importance if channel nonlinearity is important). This section begins in Section B.5.1 with a discussion of multidimensional partitioning, before then enumerating 4D and 8D coset codes in Section B.5.2.

B.5.1 Lattice Codes and Multidimensional Partitioning

Lattice Codes are sometimes construed as coset codes with G a $k \times (k + r_G)$ constant matrix (or block code). For the lattices of interest in this chapter, we need not develop the connection with block codes and instead consider $r_G = 0$ and any redundancy as part of r_Λ. For those interested in the strong connection, see the work by Forney and by Conway and Sloane. Appendix A of this chapter introduces lattices.

The fundamental gain of a lattice is defined similar to that of a coset code (where the reference uncoded system is again a Z^N lattice)

$$\gamma_f(\Lambda) \triangleq \frac{d^2_{\min}(\Lambda)}{\mathcal{V}^{2/N}(\Lambda)} = \frac{d^2_{\min}(\Lambda)}{2^{2\bar{r_\Lambda}}} \quad . \tag{B.170}$$

Multidimensional symbols are often formed by concatenation of lower-dimensional symbols. For instance, a four-dimensional code symbol is often formed from two-dimensional QAM signals as [QAM1 , QAM2], or perhaps from 4 one-dimensional PAM signals. Eight-dimensional symbols are also formed by concatenation of four two-dimensional symbols or eight one-dimensional symbols, or even two 4-dimensional symbols. Certain symbols may be allowed to follow other symbols, while certain other symbols cannot. This section attempts to enumerate and study the most common four and eight dimensional constellation lattices, and in particular their partitioning for trellis code application.

B.5.1.1 Rectangular Lattice Family

The rectangular lattice in N dimensions is just the lattice Z^N, or any coset (translation) thereof. The volume of such a lattice is

$$\mathcal{V}(Z^N) = 1 \tag{B.171}$$

and the minimum distance is

$$d_{\min} = 1 \tag{B.172}$$

leaving a fundamental lattice gain of $\gamma_f(Z^N) = 1$ or 0 dB.

Simple Lattice Constructions:
For Example B.3.2,

$$D_2 \triangleq R_2 Z^2 \quad , \tag{B.173}$$

$\mathcal{V}(D_2) = 2$ and $d_{\min}(D_2) = \sqrt{2}$, so that $\gamma_f(D_2) = 0$ dB. D_2 partitions Z^2, and $|Z^2/D_2| = 2$ so that

$$\mathcal{V}(D_2) = |Z^2/D_2| \cdot \mathcal{V}(Z^2) \quad . \tag{B.174}$$

More generally:

Theorem B.5.1 (Volume and Partitioning) *if a sublattice Λ' partitions the lattice Λ, then*

$$\mathcal{V}(\Lambda') = |\Lambda/\Lambda'| \cdot \mathcal{V}(\Lambda) \quad . \tag{B.175}$$

Proof: Because there are $\frac{1}{|\Lambda/\Lambda'|}$ as many points in Λ' as in Λ, and the union of fundamental volumes for all points in a lattice must cover N-dimensional space, then $\mathcal{V}(\Lambda')$ must be $|\Lambda/\Lambda'|$ times larger. **QED**.

Two successive applications of R_2 produce

$$R_2^2 = \left[\begin{array}{cc} 2 & 0 \\ 0 & 2 \end{array} \right] \quad , \tag{B.176}$$

a scaling of the original lattice by a factor of 2, so that $d_{\min}$ increases by 2, and volume (area) increases by a factor of 4, and γ_f remains at 0 dB. The resultant lattice is

$$R_2^2 Z^2 = R_2 D_2 = 2Z^2 \quad . \tag{B.177}$$

The semi-infinite partition chain is:

$$Z^2/R_2Z^2/R_2^2Z^2/R_2^3Z^2/R_2^4Z^2... \tag{B.178}$$

$$Z^2/D_2/2Z^2/2D_2/4Z^2/... \tag{B.179}$$

The two-dimensional partitioning for two-dimensional trellis codes is generated by successive application of the rotation operator R_2.

The concept of the rotation operator can be extended to 4 dimensions by defining

$$R_4 \triangleq \left[\begin{array}{cc} R_2 & 0 \\ 0 & R_2 \end{array} \right] \quad , \tag{B.180}$$

and multiplication of a four-dimensional lattice by R_4 amounts to applying R_2 independently to the first two coordinates of a four-dimensional lattice and to the last two coordinates of that same lattice to generate a new four-dimensional set of points. Thus,

$$R_4 Z^4 = R_2 Z^2 \otimes R_2 Z^2 \tag{B.181}$$

and that R_4Z^4 partitions Z^4 four ways, that is $|Z^4/R_4Z^4| = 4$.

$$Z^4 = R_4 Z^4 + \{[0000], [0001], [0100], [0101]\} \quad . \tag{B.182}$$

Then,

$$\mathcal{V}(R_4Z^4) = |Z^4/R_4Z^4| \ \cdot \ \mathcal{V}(Z^4) = 4 \cdot 1 = 4 \quad , \tag{B.183}$$

and that $d_{\min}^2(R_4Z^4) = 2$, so that

$$\gamma_f(R_4Z^4) = \frac{2}{4^{2/4}} = 1 \ \ (0 \text{ dB}) \quad . \tag{B.184}$$

Similarly, R_8 is

$$R_8 \triangleq \left[\begin{array}{cc} R_4 & 0 \\ 0 & R_4 \end{array} \right] \quad . \tag{B.185}$$

Then, $d_{\min}^2(R_8Z^8) = 2$, $\mathcal{V}(R_8Z^8) = 16$, and

$$\gamma_f(R_8Z^8) = \frac{2}{16^{2/8}} = 1 \ \ (0 \text{ dB}) \quad . \tag{B.186}$$

Lemma B.5.1 (Invariance of fundamental gain to squaring and rotation) *The fundamental gain of a lattice is invariant under rotation and/or squaring.*

Proof: The rotation operation increases volume by $2^{N/2}$ and squared minimum distance by 2, thus $\gamma_f(R\Lambda) = 2/\{2^{[(N/2)(2/N)]}\}\gamma_f(\Lambda) = \gamma_f(\Lambda)$. The squaring operation squares the volume and doubles the dimensionality, but does not alter the minimum distance, thus $\gamma_f(\Lambda)$ is not altered. **QED.**

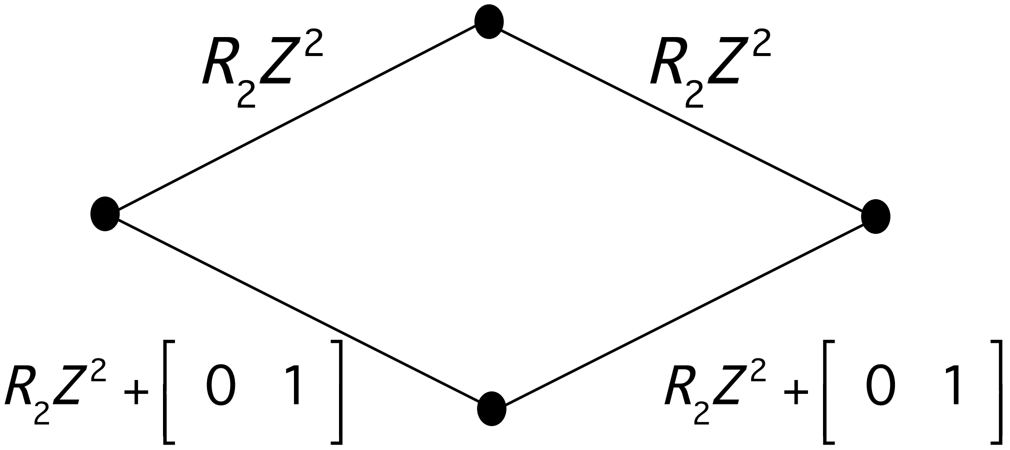


Figure B.18: Trellis for the D_4 lattice.

B.5.1.2 D - Lattice Family

The lattice $D_2 = R_2Z^2$ is a rotated (and scaled) version of the Z^2 lattice. It partitions Z^2 into two sets, D_2 and $D_2 + [0,1]$, that is

$$Z^2 = D_2 \bigcup (D_2 + [0,1]) \tag{B.187}$$

In four dimensions, that R_4Z^4 partitions Z^4 into 4 sets – the D_4 lattice partitions Z^4 into two sets. This lattice is sometimes called the "Schlafi Lattice":

$$\begin{aligned} D_4 &\stackrel{\Delta}{=} R_4Z^4 \bigcup \left(R_4Z^4 + [0,1,0,1]\right) && \text{(B.188)} \\ &= \left(R_2Z^2 \otimes R_2Z^2\right) \bigcup \left((R_2Z^2 + [0,1]) \otimes (R_2Z^2 + [0,1])\right) \quad , && \text{(B.189)} \end{aligned}$$

which can be identified as all those points in Z^4 that have even squared norms. Thus, D_4 partitions Z^4 into two sets of points (evens and odds). For D_4, $d^2_{\text{min}}(D_4) = 2$ and $\mathcal{V}(D_4) = 2$, so that

$$\gamma_f(D_4) = \frac{2}{2^{2/4}} = \sqrt{2} \ = 1.5 \text{ dB} \quad . \tag{B.190}$$

Equation (B.190) relates that the D_4 lattice is better in terms of packing points per unit volume than the rectangular lattice family by a factor of 1.5 dB. In fact, D_4 is the best such **lattice** in four dimensions.

The D_4 lattice is a simple coset code or lattice code that can be described by a trellis that starts and ends with a single state, but which has two states in the middle as shown in Figure B.18. The Viterbi Algorithm for MLSD can be applied to the trellis in Figure B.18 to decode the D_4 lattice in the same way that the Viterbi Algorithm is used for trellis codes.

In eight dimensions, R_8Z^8 partitions Z^8 into 16 cosets of R_8Z^8 so that

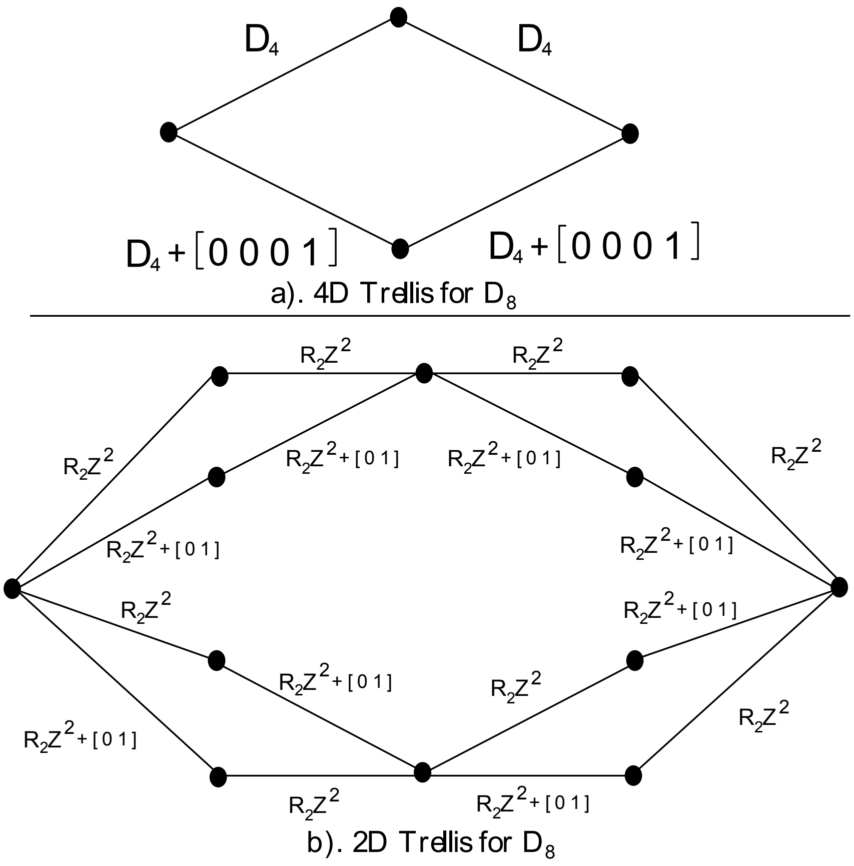
$$|Z^8/R_8Z^8| = 16 \quad . \tag{B.191}$$

Also since $\left|Z^4/D_4\right| = 2$, then

$$|Z^8/(D_4)^2| = 4 \quad . \tag{B.192}$$

A binary partition (of order 2) is desirable:

$$D_8 \stackrel{\Delta}{=} (D_4 \otimes D_4) \bigcup ((D_4 + [0,0,0,1]) \otimes (D_4 + [0,0,0,1])) \quad , \tag{B.193}$$



a). 4D Trellis for D_8

b). 2D Trellis for D_8

Figure B.19: Trellis for the D_8 lattice.

which is the subset of Z^8 of all points with even squared norms. Thus,

$$Z^8 = D_8 \bigcup (D_8 + [0,0,0,0,0,0,0,1]) \quad . \tag{B.194}$$

$d^2_{\text{min}}(D_8) = 2$ and $\mathcal{V}(D_8) = 2$, so that

$$\gamma_f(D_8) = \frac{2}{2^{2/8}} = 2^{.75} \quad = 2.27 \text{ dB} \quad . \tag{B.195}$$

Thus, D_8 is somewhat better than D_4. Also observe that $(D_4)^2$ partitions D_8 from (B.193), that $|D_8/(D_4)^2| = 2$, and that $\gamma_f(D_4^2) = 1.5$ dB, which also follows from Lemma B.5.1

A trellis similar to that in Figure B.18 can be drawn for the D_8 lattice as shown in Figure B.19. Each of the D_4 trellises in Figure B.19 can be replaced by D_4 trellises leading to the more informative 2-dimensional trellis for D_8 in Figure B.19. Again, the Viterbi algorithm can decode the D_8 lattice using these trellises.

B.5.1.3 The DE_8 Lattice

The DE_8 lattice is defined by

$$DE_8 \triangleq (R_4Z^4)^2 \bigcup \left((R_4Z^4)^2 + [0,1,0,1,0,1,0,1]\right) \tag{B.196}$$

$$= R_8Z^8 \bigcup \left(R_8Z^8 + [0,1,0,1,0,1,0,1]\right) \tag{B.197}$$

DE_8 partitions $(D_4)^2$ into two groups by

$$\begin{aligned}(D_4)^2 &= \left[\left(R_4Z^4\right) \bigcup \left(R_4Z^4 + [0,1,0,1]\right)\right]^2 \\ &= \left(R_4Z^4\right)^2 \bigcup \left(R_4Z^4 + [0,1,0,1]\right)^2 \bigcup \left(R_4Z^4 \otimes \left(R_4Z^4 + [0,1,0,1]\right)\right) \bigcup \left(\left(R_4Z^4 + [0,1,0,1]\right) \otimes R_4Z^4\right) \\ &= DE_8 \bigcup \left(DE_8 + [0,1,0,1,0,0,0,0]\right) \quad . \qquad \text{(B.198)}\end{aligned}$$

The last step notes that with respect to the R_4Z^4 lattice, adding [0,2,0,2] is equivalent to adding [0,0,0,0]. $d^2_{\min}(DE_8) = 2$ and $\mathcal{V}(DE_8) = 8$, so that

$$\gamma_f(DE_8) = \frac{2}{8^{2/8}} = 2^{.25} \quad = .73 \text{ dB} \quad . \qquad \text{(B.199)}$$

So far, previous results have established the partition chain

$$Z^8/D_8/D_4^2/DE_8 \qquad \text{(B.200)}$$

with a factor of 2 increase in lattice fundamental volume at each step in the partitioning. One is tempted to complete the chain by partitioning again into R_8Z^8, which would be a valid partition. The next subsection shows another partition into a much better 8 dimensional lattice.

The trellis diagram for the DE_8 lattice is trivial and left to the reader as an exercise.

B.5.1.4 The Gosset (E_8) Lattice

The Gosset or E_8 Lattice is the most dense **lattice** in 8 dimensions. It is defined by

$$E_8 \stackrel{\Delta}{=} R_8D_8 \bigcup \left(R_8D_8 + [0,1,0,1,0,1,0,1]\right) \quad . \qquad \text{(B.201)}$$

All norms of points in E_8 are integer multiples of 4. Since rotation by R_N increases distance by a factor of $\sqrt{2}$, inspection of the coset leader in the second term of (B.201) leads to

$$d^2_{\min}(E_8) = 4 \qquad \text{(B.202)}$$

Further, $|E_8/R_8D_8| = 2$, so that

$$\mathcal{V}(E_8) = \frac{1}{2}\mathcal{V}(R_8D_8) = \frac{1}{2}16 \cdot 2 = 16 \quad . \qquad \text{(B.203)}$$

Then,

$$\gamma_f(E_8) = \frac{4}{16^{2/8}} = 2 \ \ (3 \text{ dB}) \quad . \qquad \text{(B.204)}$$

A simple trellis in terms of the 4 dimensional constituents appears in Figure B.20, where the Cartesian product decomposition for D_8 in Equation (B.193) has been used along with the fact that rotation by R_8 is the same as rotating each four-dimensional constituent by R_4. A two-dimensional trellis is then constructed by drawing the 2D trellises for D_4 wherever they appear in Figure B.20. This two-dimensional trellis appears in Figure B.21.

The assumption that E_8 partitions DE_8 is justified by taking

$$Z^8 = D_8 \bigcup \left(D_8 + [0,0,0,0,0,0,0,1]\right) \qquad \text{(B.205)}$$

and premultiplying by R_8

$$R_8Z^8 = R_8D_8 \bigcup \left(R_8D_8 + [0,0,0,0,0,0,1,-1]\right) \quad , \qquad \text{(B.206)}$$

making (B.197) the same as

$$\begin{aligned} DE_8 &= R_8D_8 \bigcup \left(R_8D_8 + [0,0,0,0,0,0,1,-1]\right) & \text{(B.207)} \\ & \bigcup \left(R_8D_8 + [0,1,0,1,0,1,0,1]\right) \bigcup \left(R_8D_8 + [0,1,0,1,0,1,1,0]\right) & \text{(B.208)} \\ &= E_8 \bigcup \left(E_8 + [0,0,0,0,0,0,1,-1]\right) \quad . & \text{(B.209)}\end{aligned}$$

Thus, E_8 partitions DE_8 and $|DE_8/E_8| = 2$.

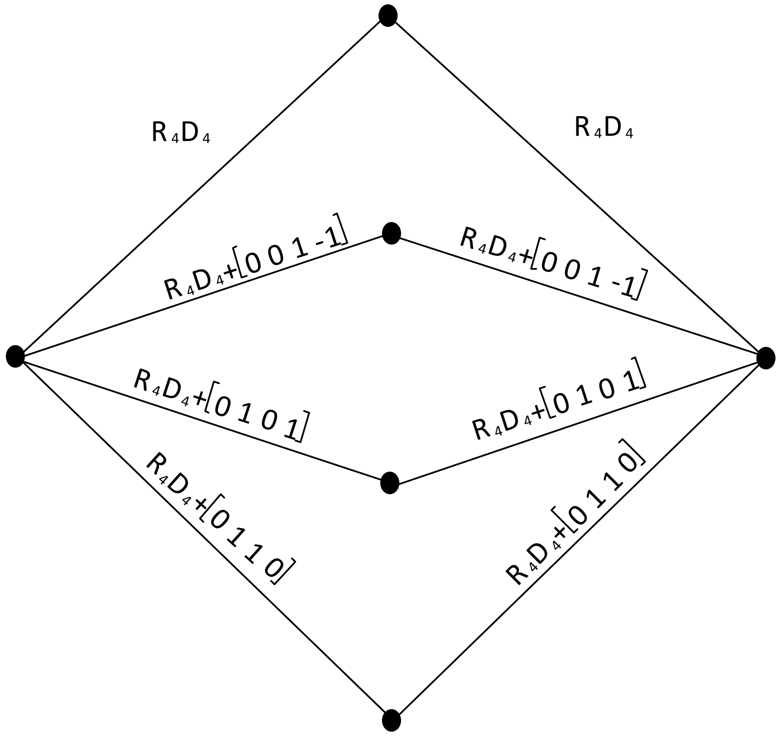


Figure B.20: 4D Trellis for the E_8 lattice.

v_i		$d^2_{\min}(\Lambda)$	$\mathcal{V}(\Lambda)$	$\gamma_f(\Lambda)$ dB
$-$	Z^4	1	1	0
v_0	D_4	2	2	1.5
v_1	R_4Z^4	2	4	0
v_2	R_4D_4	4	8	1.5
v_3	$2Z^4$	4	16	0
v_4	$2D_4$	8	32	1.5

Table B.7: Four-dimensional partition tower and lattice parameters.

B.5.1.5 4 and 8 Dimensional Partition Chains

The previous subsections established the four-dimensional partition chain

$$Z^4/D_4/R_4Z^4/R_4D_4/2Z^4/2D_4/... \qquad \text{(B.210)}$$

and the eight-dimensional partition chain

$$Z^8/D_8/D_4^2/DE_8/E_8/R_8D_8/(R_4D_4)^2/R_8DE_8/R_8E_8/... \qquad \text{(B.211)}$$

These partition chains are summarized in the partition towers and accompanying tables in Tables B.7 and B.8. These partitionings are also used extensively in four- and eight-dimensional trellis codes, as discussed in the next section. Figure B.22 is a partition tree showing the specific labels for a four-dimensional partitioning with Ungerboeck labeling. Each of the cosets of the original constellation in Figure B.22 can be written as (possibly unions of) Cartesian products of two-dimensional cosets in the

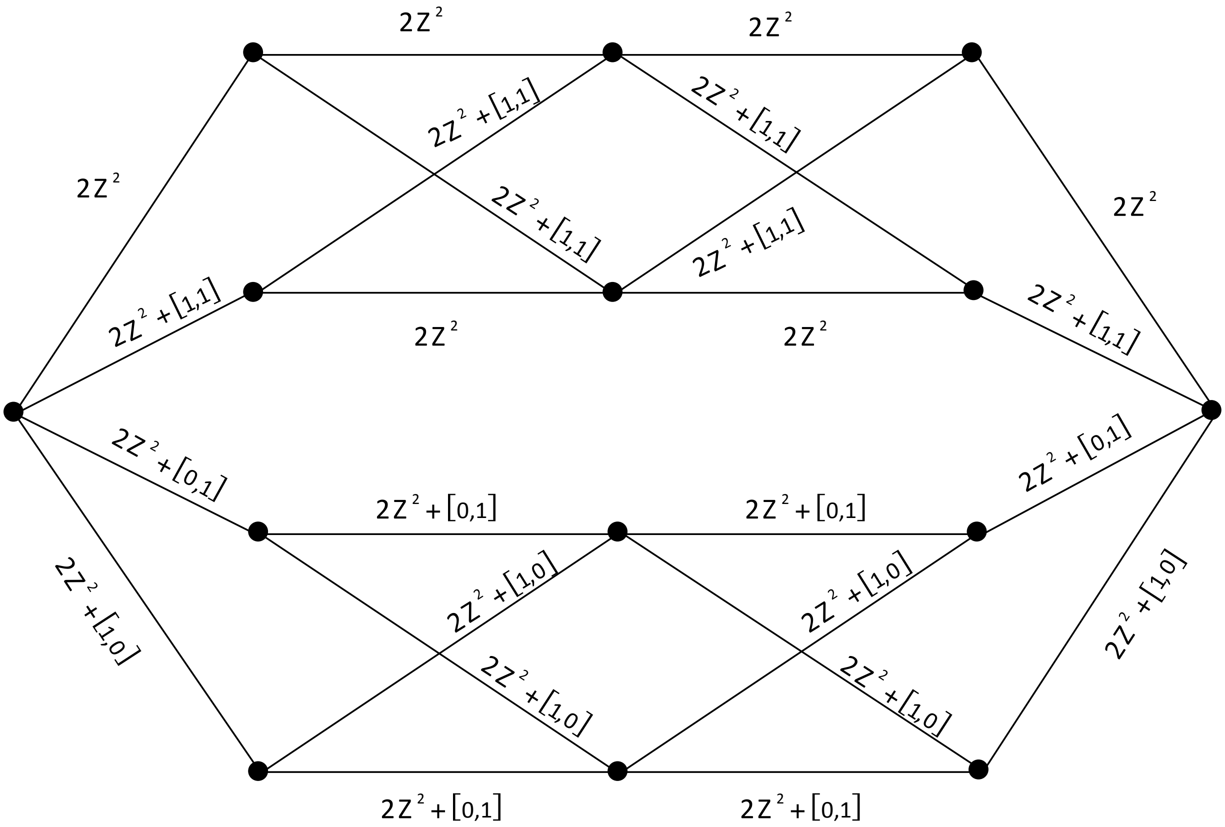


Figure B.21: 2D Trellis for the E_8 lattice.

v_i		$d^2_{\min}(\Lambda)$	$\mathcal{V}(\Lambda)$	$\gamma_f(\Lambda)$ dB
−	Z^8	1	1	0
v_0	D_8	2	2	2.27
v_1	$(D_4)^2$	2	4	1.5
v_2	DE_8	2	8	.73
v_3	E_8	4	16	3
v_4	R_8D_8	4	32	2.27
v_5	$R_8(D_4)^2$	4	64	1.5
v_6	R_8DE_8	4	128	.73
v_7	R_8E_8	8	256	3

Table B.8: Eight-Dimensional Partition Tower and Lattice Parameters

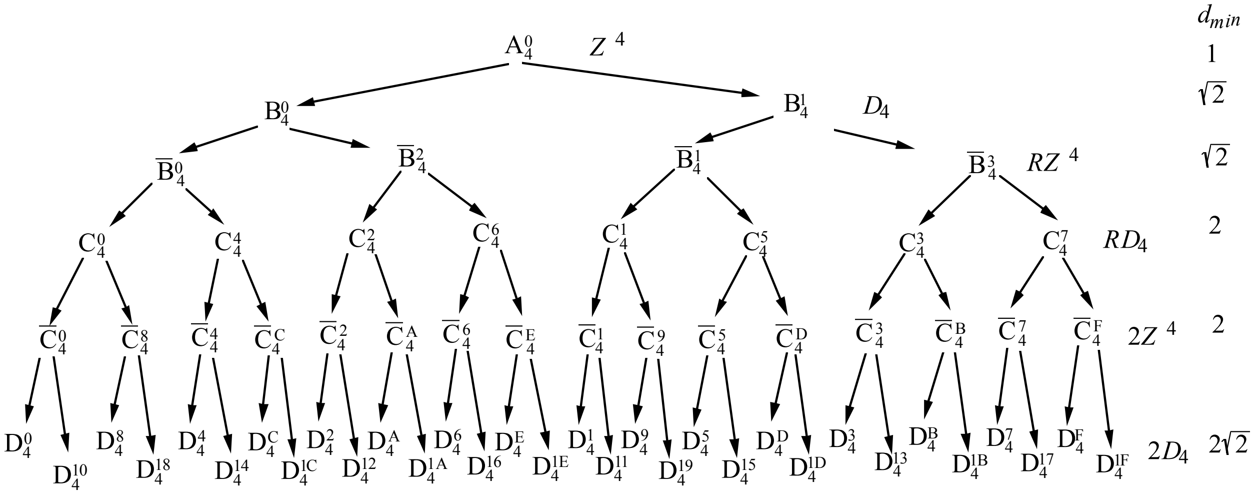


Figure B.22: Four-dimensional partition tree with Ungerboeck labeling - indices of $\bar{C}$ and of D are in hexadecimal.

partitioning of the Z^2 lattice. This section lists those Cartesian products for reference:

$$A_4^0 = A_2^0 \otimes A_2^0 \tag{B.212}$$

$$B_4^0 = \left(B_2^0 \otimes B_2^0\right) \bigcup \left(B_2^1 \otimes B_2^1\right) \tag{B.213}$$

$$B_4^1 = \left(B_2^0 \otimes B_2^1\right) \bigcup \left(B_2^1 \otimes B_2^0\right) \tag{B.214}$$

$$\bar{B}_4^0 = B_2^0 \otimes B_2^0 \tag{B.215}$$

$$\bar{B}_4^2 = B_2^1 \otimes B_2^1 \tag{B.216}$$

$$\bar{B}_4^1 = B_2^0 \otimes B_2^1 \tag{B.217}$$

$$\bar{B}_4^3 = B_2^1 \otimes B_2^0 \tag{B.218}$$

$$C_4^0 = \left(C_2^0 \otimes C_2^0\right) \bigcup \left(C_2^2 \otimes C_2^2\right) \tag{B.219}$$

$$C_4^4 = \left(C_2^0 \otimes C_2^2\right) \bigcup \left(C_2^2 \otimes C_2^0\right) \tag{B.220}$$

$$C_4^2 = \left(C_2^1 \otimes C_2^1\right) \bigcup \left(C_2^3 \otimes C_2^3\right) \tag{B.221}$$

$$C_4^6 = \left(C_2^1 \otimes C_2^3\right) \bigcup \left(C_2^3 \otimes C_2^1\right) \tag{B.222}$$

$$C_4^1 = \left(C_2^0 \otimes C_2^1\right) \bigcup \left(C_2^2 \otimes C_2^3\right) \tag{B.223}$$

$$C_4^5 = \left(C_2^0 \otimes C_2^3\right) \bigcup \left(C_2^2 \otimes C_2^1\right) \tag{B.224}$$

$$C_4^3 = \left(C_2^1 \otimes C_2^0\right) \bigcup \left(C_2^3 \otimes C_2^2\right) \tag{B.225}$$

$$C_4^7 = \left(C_2^1 \otimes C_2^2\right) \bigcup \left(C_2^3 \otimes C_2^0\right) \tag{B.226}$$

$$\bar{C}_4^0 = C_2^0 \otimes C_2^0 \tag{B.227}$$

$$\bar{C}_4^8 = C_2^2 \otimes C_2^2 \tag{B.228}$$

$$\bar{C}_4^4 = C_2^0 \otimes C_2^2 \tag{B.229}$$

$$\bar{C}_4^C = C_2^2 \otimes C_2^0 \tag{B.230}$$

$$\bar{C}_4^2 = C_2^1 \otimes C_2^1 \tag{B.231}$$

$$\bar{C}_4^A = C_2^3 \otimes C_2^3 \tag{B.232}$$

$$\bar{C}_4^6 = C_2^1 \otimes C_2^3 \tag{B.233}$$

$$\bar{C}_4^E = C_2^3 \otimes C_2^1 \tag{B.234}$$

$$\bar{C}_4^1 = C_2^0 \otimes C_2^1 \tag{B.235}$$

$$\bar{C}_4^9 = C_2^2 \otimes C_2^3 \tag{B.236}$$

$$\bar{C}_4^5 = C_2^0 \otimes C_2^3 \tag{B.237}$$

$$\bar{C}_4^D = C_2^2 \otimes C_2^1 \tag{B.238}$$

$$\bar{C}_4^3 = C_2^1 \otimes C_2^0 \tag{B.239}$$

$$\bar{C}_4^B = C_2^3 \otimes C_2^2 \tag{B.240}$$

$$\bar{C}_4^7 = C_2^1 \otimes C_2^2 \tag{B.241}$$

$$\bar{C}_4^F = C_2^3 \otimes C_2^0 \tag{B.242}$$

$$D_4^0 = \left(D_2^0 \otimes D_2^0\right) \bigcup \left(D_2^4 \otimes D_2^4\right) \tag{B.243}$$

$$D_4^{10} = \left(D_2^0 \otimes D_2^4\right) \bigcup \left(D_2^4 \otimes D_2^0\right) \tag{B.244}$$

$$D_4^8 = \left(D_2^2 \otimes D_2^2\right) \bigcup \left(D_2^6 \otimes D_2^6\right) \tag{B.245}$$

$$D_4^{18} = \left(D_2^2 \otimes D_2^6\right) \bigcup \left(D_2^6 \otimes D_2^2\right) \tag{B.246}$$

$$D_4^4 = \left(D_2^0 \otimes D_2^2\right) \bigcup \left(D_2^4 \otimes D_2^6\right) \tag{B.247}$$

$$D_4^{14} = \left(D_2^0 \otimes D_2^6\right) \bigcup \left(D_2^4 \otimes D_2^2\right) \tag{B.248}$$

$$D_4^C = \left(D_2^2 \otimes D_2^0\right) \bigcup \left(D_2^6 \otimes D_2^4\right) \tag{B.249}$$

$$D_4^{1C} = \left(D_2^2 \otimes D_2^4\right) \bigcup \left(D_2^6 \otimes D_2^0\right) \tag{B.250}$$

$$D_4^2 = \left(D_2^1 \otimes D_2^1\right) \bigcup \left(D_2^5 \otimes D_2^5\right) \tag{B.251}$$

$$D_4^{12} = \left(D_2^1 \otimes D_2^5\right) \bigcup \left(D_2^5 \otimes D_2^1\right) \tag{B.252}$$

$$D_4^A = \left(D_2^3 \otimes D_2^3\right) \bigcup \left(D_2^7 \otimes D_2^7\right) \tag{B.253}$$

$$D_4^{1A} = \left(D_2^3 \otimes D_2^7\right) \bigcup \left(D_2^7 \otimes D_2^3\right) \tag{B.254}$$

$$D_4^6 = \left(D_2^1 \otimes D_2^3\right) \bigcup \left(D_2^5 \otimes D_2^7\right) \tag{B.255}$$

$$D_4^{16} = \left(D_2^1 \otimes D_2^7\right) \bigcup \left(D_2^5 \otimes D_2^3\right) \tag{B.256}$$

$$D_4^E = \left(D_2^3 \otimes D_2^1\right) \bigcup \left(D_2^7 \otimes D_2^5\right) \tag{B.257}$$

$$D_4^{1E} = \left(D_2^3 \otimes D_2^5\right) \bigcup \left(D_2^7 \otimes D_2^1\right) \tag{B.258}$$

$$D_4^1 = \left(D_2^0 \otimes D_2^1\right) \bigcup \left(D_2^4 \otimes D_2^5\right) \tag{B.259}$$

$$D_4^{11} = \left(D_2^0 \otimes D_2^5\right) \bigcup \left(D_2^4 \otimes D_2^1\right) \tag{B.260}$$

$$D_4^9 = \left(D_2^2 \otimes D_2^3\right) \bigcup \left(D_2^6 \otimes D_2^7\right) \tag{B.261}$$

$$D_4^{19} = \left(D_2^2 \otimes D_2^7\right) \bigcup \left(D_2^6 \otimes D_2^3\right) \tag{B.262}$$

$$D_4^5 = \left(D_2^0 \otimes D_2^3\right) \bigcup \left(D_2^4 \otimes D_2^7\right) \tag{B.263}$$

$$D_4^{15} = \left(D_2^0 \otimes D_2^7\right) \bigcup \left(D_2^4 \otimes D_2^3\right) \tag{B.264}$$

$$D_4^D = \left(D_2^2 \otimes D_2^1\right) \bigcup \left(D_2^6 \otimes D_2^5\right) \tag{B.265}$$

$$D_4^{1D} = \left(D_2^2 \otimes D_2^5\right) \bigcup \left(D_2^6 \otimes D_2^1\right) \tag{B.266}$$

$$D_4^3 = \left(D_2^1 \otimes D_2^0\right) \bigcup \left(D_2^5 \otimes D_2^4\right) \tag{B.267}$$

$$D_4^{13} = \left(D_2^1 \otimes D_2^4\right) \bigcup \left(D_2^5 \otimes D_2^0\right) \tag{B.268}$$

$$D_4^B = \left(D_2^3 \otimes D_2^2\right) \bigcup \left(D_2^7 \otimes D_2^6\right) \tag{B.269}$$

$$D_4^{1B} = \left(D_2^3 \otimes D_2^6\right) \bigcup \left(D_2^7 \otimes D_2^2\right) \tag{B.270}$$

$$D_4^7 = \left(D_2^1 \otimes D_2^2\right) \bigcup \left(D_2^5 \otimes D_2^6\right) \tag{B.271}$$

$$D_4^{17} = \left(D_2^1 \otimes D_2^6\right) \bigcup \left(D_2^5 \otimes D_2^2\right) \tag{B.272}$$

$$D_4^F = \left(D_2^3 \otimes D_2^0\right) \bigcup \left(D_2^7 \otimes D_2^4\right) \tag{B.273}$$

$$D_4^{1F} = \left(D_2^3 \otimes D_2^4\right) \bigcup \left(D_2^7 \otimes D_2^0\right) \tag{B.274}$$

In order to more completely understand the entries in the four-dimensional coding tables, we also summarize some partitioning results in four-dimensions in Table B.9.

	A_4^0							
$d_{\min}$ w.r.t. A_4^0	1							
N_e w.r.t. A_4^0	8							
	B_4^0				B_4^1			
$d_{\min}$ w.r.t. B_4^0	$\sqrt{2}$				1			
N_e w.r.t. B_4^0	24				8			
	$\bar{B}_4^0$		$\bar{B}_4^2$		$\bar{B}_4^1$		$\bar{B}_4^3$	
$d_{\min}$ w.r.t. $\bar{B}_4^0$	$\sqrt{2}$		$\sqrt{2}$		1		1	
N_e w.r.t. $\bar{B}_4^0$	8		16		4		4	
	C_4^0	C_4^4	C_4^2	C_4^6	C_4^1	C_4^5	C_4^3	C_4^7
$d_{\min}$ w.r.t. C_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. C_4^0	24	8	8	8	2	2	2	2
	$\bar{C}_4^0$	$\bar{C}_4^4$	$\bar{C}_4^2$	$\bar{C}_4^6$	$\bar{C}_4^1$	$\bar{C}_4^5$	$\bar{C}_4^3$	$\bar{C}_4^7$
$d_{\min}$ w.r.t. $\bar{C}_4^0$	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. $\bar{C}_4^0$	8	4	4	4	2	2	2	8
	$\bar{C}_4^8$	$\bar{C}_4^C$	$\bar{C}_4^A$	$\bar{C}_4^E$	$\bar{C}_4^9$	$\bar{C}_4^D$	$\bar{C}_4^B$	$\bar{C}_4^F$
$d_{\min}$ w.r.t. $\bar{C}_4^0$	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. $\bar{C}_4^0$	16	4	4	4	8	8	8	2
	D_4^0	D_4^4	D_4^2	D_4^6	D_4^1	D_4^5	D_4^3	D_4^7
$d_{\min}$ w.r.t. D_4^0	$2\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. D_4^0	24	2	2	2	1	1	1	4
	D_4^8	D_4^C	D_4^A	D_4^E	D_4^9	D_4^D	D_4^B	D_4^F
$d_{\min}$ w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. D_4^0	8	2	2	2	4	4	4	1
	D_4^{10}	D_4^{14}	D_4^{12}	D_4^{16}	D_4^{11}	D_4^{15}	D_4^{13}	D_4^{17}
$d_{\min}$ w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. D_4^0	8	2	2	2	1	1	1	4
	D_4^{18}	D_4^{1C}	D_4^{1A}	D_4^{1E}	D_4^{19}	D_4^{1D}	D_4^{1B}	D_4^{1F}
$d_{\min}$ w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. D_4^0	8	2	2	2	4	4	4	1

Table B.9: Four-Dimensional Partitioning

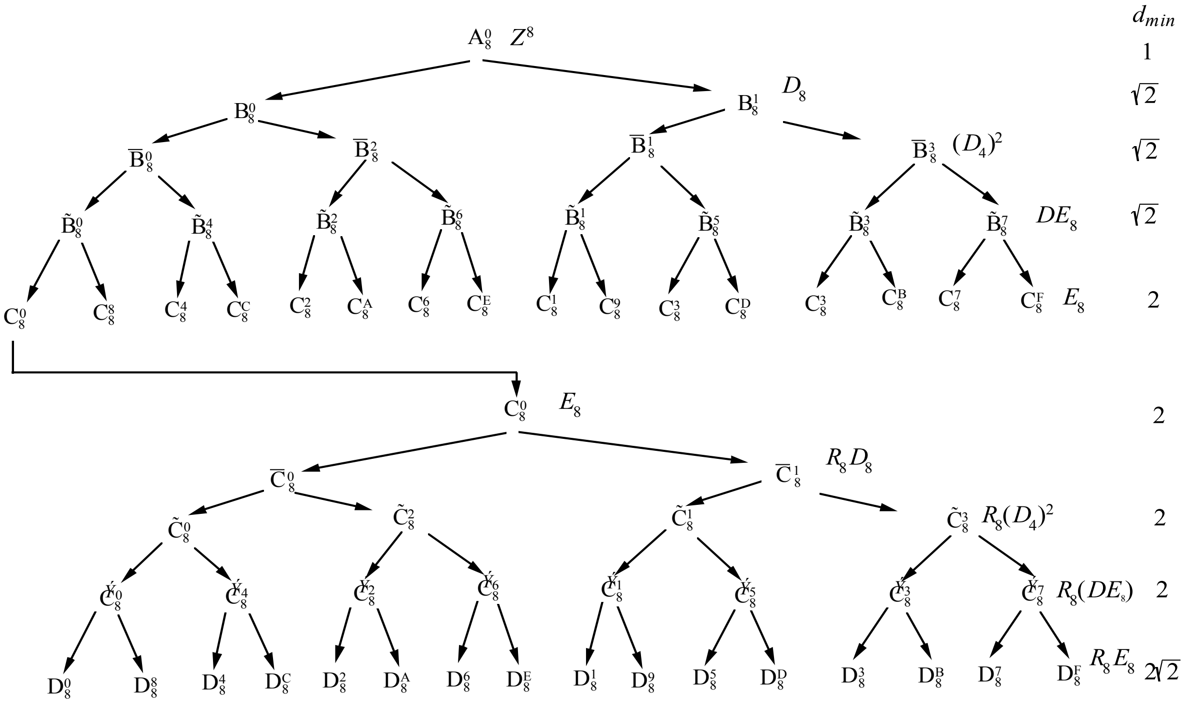

d_{min}
A_8^0 Z^8
1
B_8^0
B_8^1 D_8
$\sqrt{2}$
$\bar{B}_8^0$
$\bar{B}_8^2$
$\bar{B}_8^1$
$\bar{B}_8^3$ $(D_4)^2$
$\sqrt{2}$
$\tilde{B}_8^0$
$\tilde{B}_8^4$
$\tilde{B}_8^2$
$\tilde{B}_8^6$
$\tilde{B}_8^1$
$\tilde{B}_8^5$
$\tilde{B}_8^3$
$\tilde{B}_8^7$ DE_8
$\sqrt{2}$
C_8^0 C_8^8 C_8^4 C_8^C C_8^2 C_8^A C_8^6 C_8^E C_8^1 C_8^9 C_8^3 C_8^D C_8^3 C_8^B C_8^7 C_8^F E_8
2
C_8^0 E_8
2
$\bar{C}_8^0$
$\bar{C}_8^1$ R_8D_8
$\tilde{C}_8^0$
$\tilde{C}_8^2$
$\tilde{C}_8^1$
$\tilde{C}_8^3$ $R_8(D_4)^2$
2
C_8^0 C_8^4 C_8^2 C_8^6 C_8^1 C_8^5 C_8^3 C_8^7 $R_8(DE_8)$
2
D_8^0 D_8^8 D_8^4 D_8^C D_8^2 D_8^A D_8^6 D_8^E D_8^1 D_8^9 D_8^5 D_8^D D_8^3 D_8^B D_8^7 D_8^F R_8E_8
$2\sqrt{2}$

Figure B.23: Eight-dimensional partition tree with Ungerboeck labeling.

The subscript on a partition refers to the dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck labeling.

Figure B.23 is a partition tree showing the specific labels for a eight-dimensional partitioning with Ungerboeck labeling. Each of the cosets of the original constellation in Figure B.23 can be written as (possibly unions of) Cartesian products of two-dimensional cosets in the partitioning of the Z^4 lattice. Those Cartesian products are (for reference):

$$A_8^0 = A_4^0 \otimes A_4^0 \tag{B.275}$$

$$B_8^0 = \left(B_4^0 \otimes B_4^0\right) \bigcup \left(B_4^1 \otimes B_4^1\right) \tag{B.276}$$

$$B_8^1 = \left(B_4^0 \otimes B_4^1\right) \bigcup \left(B_4^1 \otimes B_4^0\right) \tag{B.277}$$

$$\bar{B}_8^0 = B_4^0 \otimes B_4^0 \tag{B.278}$$

$$\bar{B}_8^2 = B_4^1 \otimes B_4^1 \tag{B.279}$$

$$\bar{B}_8^1 = B_4^0 \otimes B_4^1 \tag{B.280}$$

$$\bar{B}_8^3 = B_4^1 \otimes B_4^0 \tag{B.281}$$

$$\begin{aligned}
\tilde{B}_8^0 &= \left(\bar{B}_4^0 \otimes \bar{B}_4^0\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^2\right) && \text{(B.282)}\\
\tilde{B}_8^4 &= \left(\bar{B}_4^0 \otimes \bar{B}_4^2\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^0\right) && \text{(B.283)}\\
\tilde{B}_8^2 &= \left(\bar{B}_4^1 \otimes \bar{B}_4^1\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^3\right) && \text{(B.284)}\\
\tilde{B}_8^6 &= \left(\bar{B}_4^1 \otimes \bar{B}_4^3\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^1\right) && \text{(B.285)}\\
\tilde{B}_8^1 &= \left(\bar{B}_4^0 \otimes \bar{B}_4^1\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^3\right) && \text{(B.286)}\\
\tilde{B}_8^5 &= \left(\bar{B}_4^0 \otimes \bar{B}_4^3\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^1\right) && \text{(B.287)}\\
\tilde{B}_8^3 &= \left(\bar{B}_4^1 \otimes \bar{B}_4^0\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^2\right) && \text{(B.288)}\\
\tilde{B}_8^7 &= \left(\bar{B}_4^1 \otimes \bar{B}_4^2\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^0\right) && \text{(B.289)}
\end{aligned}$$

$$\begin{aligned}
C_8^0 &= \left(C_4^0 \otimes C_4^0\right) \bigcup \left(C_4^4 \otimes C_4^4\right) \bigcup \left(C_4^2 \otimes C_4^2\right) \bigcup \left(C_4^6 \otimes C_4^6\right)\\
C_8^8 &= \left(C_4^0 \otimes C_4^4\right) \bigcup \left(C_4^4 \otimes C_4^0\right) \bigcup \left(C_4^2 \otimes C_4^6\right) \bigcup \left(C_4^6 \otimes C_4^2\right)\\
C_8^4 &= \left(C_4^0 \otimes C_4^2\right) \bigcup \left(C_4^4 \otimes C_4^6\right) \bigcup \left(C_4^2 \otimes C_4^0\right) \bigcup \left(C_4^6 \otimes C_4^4\right)\\
C_8^C &= \left(C_4^0 \otimes C_4^6\right) \bigcup \left(C_4^4 \otimes C_4^2\right) \bigcup \left(C_4^2 \otimes C_4^4\right) \bigcup \left(C_4^6 \otimes C_4^0\right)\\
C_8^2 &= \left(C_4^1 \otimes C_4^1\right) \bigcup \left(C_4^5 \otimes C_4^5\right) \bigcup \left(C_4^3 \otimes C_4^3\right) \bigcup \left(C_4^7 \otimes C_4^7\right)\\
C_8^A &= \left(C_4^1 \otimes C_4^5\right) \bigcup \left(C_4^5 \otimes C_4^1\right) \bigcup \left(C_4^3 \otimes C_4^7\right) \bigcup \left(C_4^7 \otimes C_4^3\right)\\
C_8^6 &= \left(C_4^1 \otimes C_4^3\right) \bigcup \left(C_4^5 \otimes C_4^7\right) \bigcup \left(C_4^3 \otimes C_4^1\right) \bigcup \left(C_4^7 \otimes C_4^5\right)\\
C_8^E &= \left(C_4^1 \otimes C_4^7\right) \bigcup \left(C_4^5 \otimes C_4^3\right) \bigcup \left(C_4^3 \otimes C_4^5\right) \bigcup \left(C_4^7 \otimes C_4^1\right)\\
C_8^1 &= \left(C_4^0 \otimes C_4^1\right) \bigcup \left(C_4^4 \otimes C_4^5\right) \bigcup \left(C_4^2 \otimes C_4^3\right) \bigcup \left(C_4^6 \otimes C_4^7\right)\\
C_8^9 &= \left(C_4^0 \otimes C_4^5\right) \bigcup \left(C_4^4 \otimes C_4^1\right) \bigcup \left(C_4^2 \otimes C_4^7\right) \bigcup \left(C_4^6 \otimes C_4^3\right)\\
C_8^5 &= \left(C_4^0 \otimes C_4^3\right) \bigcup \left(C_4^4 \otimes C_4^7\right) \bigcup \left(C_4^2 \otimes C_4^1\right) \bigcup \left(C_4^6 \otimes C_4^5\right)\\
C_8^D &= \left(C_4^0 \otimes C_4^7\right) \bigcup \left(C_4^4 \otimes C_4^3\right) \bigcup \left(C_4^2 \otimes C_4^5\right) \bigcup \left(C_4^6 \otimes C_4^1\right)\\
C_8^3 &= \left(C_4^1 \otimes C_4^0\right) \bigcup \left(C_4^5 \otimes C_4^4\right) \bigcup \left(C_4^3 \otimes C_4^2\right) \bigcup \left(C_4^7 \otimes C_4^6\right)\\
C_8^B &= \left(C_4^1 \otimes C_4^4\right) \bigcup \left(C_4^5 \otimes C_4^0\right) \bigcup \left(C_4^3 \otimes C_4^6\right) \bigcup \left(C_4^7 \otimes C_4^2\right)\\
C_8^7 &= \left(C_4^1 \otimes C_4^2\right) \bigcup \left(C_4^5 \otimes C_4^6\right) \bigcup \left(C_4^3 \otimes C_4^0\right) \bigcup \left(C_4^7 \otimes C_4^4\right)\\
C_8^F &= \left(C_4^1 \otimes C_4^6\right) \bigcup \left(C_4^5 \otimes C_4^2\right) \bigcup \left(C_4^3 \otimes C_4^4\right) \bigcup \left(C_4^7 \otimes C_4^0\right)
\end{aligned}$$

For the reader's and designer's assistance, Table B.10 lists the distances and nearest neighbor counts for eight-dimensional partitioning

B.5.2 Multidimensional Trellis Codes

This section returns to the coset-code encoder, which is re-illustrated in Figure B.24. Typically, $r_G = 1$, although there are a few (mainly impractical) codes for which $r_G > 1$. Thus, signal expansion is over many dimensions, and thus there is a smaller constellation expansion over any particular dimension. However, as determined in Section B.5.1, more levels of partitioning will be necessary to increase distance on the parallel transitions defined by Λ' and its cosets to an attractive level. The lower signal expansion per dimension is probably the most attractive practical feature of multi-dimensional codes. Constellation expansion makes the coded signals more susceptible to channel nonlinearity and carrier jitter in QAM. Constellation expansion can also render decision-directed (on a symbol-by-symbol basis)

	A_8^0							
$d_{\min}$ w.r.t. A_8^0	1							
N_e w.r.t. A_8^0	16							
	B_8^0				B_8^1			
$d_{\min}$ w.r.t. B_8^0	$\sqrt{2}$				1			
N_e w.r.t. B_8^0	132				16			
	$\bar{B}_8^0$		$\bar{B}_8^2$		$\bar{B}_8^1$		$\bar{B}_8^3$	
$d_{\min}$ w.r.t. $\bar{B}_8^0$	$\sqrt{2}$		$\sqrt{2}$		1		1	
N_e w.r.t. $\bar{B}_8^0$	48		64		8		8	
	$\tilde{B}_8^0$	$\tilde{B}_8^4$	$\tilde{B}_8^2$	$\tilde{B}_8^6$	$\tilde{B}_8^1$	$\tilde{B}_8^5$	$\tilde{B}_8^3$	$\tilde{B}_8^7$
$d_{\min}$ w.r.t. $\tilde{B}_8^0$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. $\tilde{B}_8^0$	16	32	32	32	4	4	4	4
	C_8^0	C_8^4	C_8^2	C_8^6	C_8^1	C_8^5	C_8^3	C_8^7
$d_{\min}$ w.r.t. C_8^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. C_8^0	240	16	16	16	2	2	2	2
	C_8^8	C_8^C	C_8^A	C_8^E	C_8^9	C_8^D	C_8^B	C_8^F
$d_{\min}$ w.r.t. C_8^0	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. C_8^0	16	16	16	16	2	2	2	2

Table B.10: Eight-Dimensional Partitioning

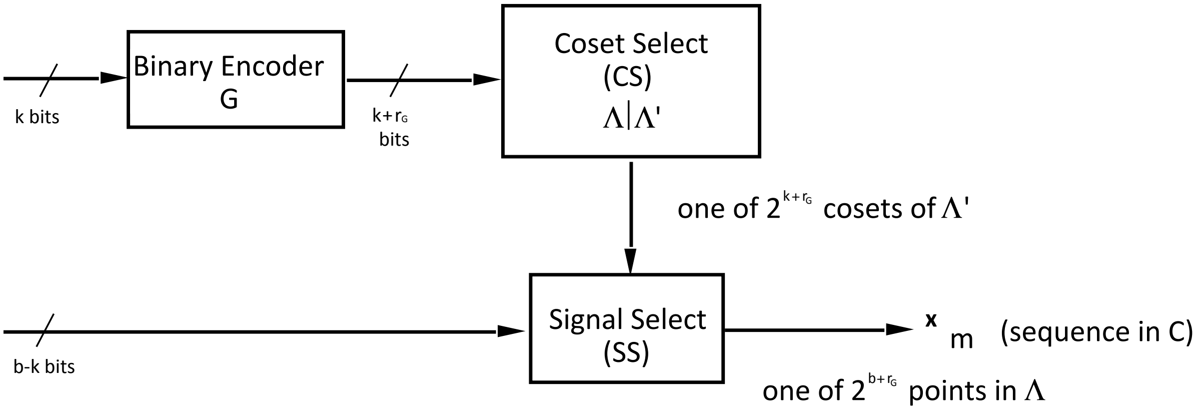


Figure B.24: The coset-code encoder.

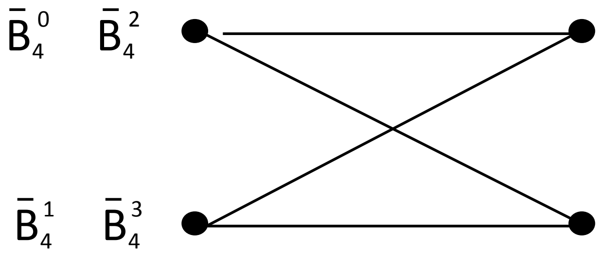


Figure B.25: Trellis for 2-state, rate 1/2, 4D code.

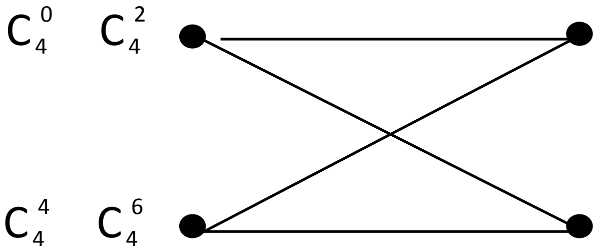


Figure B.26: Trellis for 2-state, rate 1/2, 4D Code, based on D_4.

timing and carrier loops, as well as the decision-feedback equalizer, sub-desirable in their performance due to increased symbol-by-symbol error rates.

Multidimensional codes can be attractive because the computation required to implement the Viterbi Detector is distributed over a longer time interval, usually resulting in a slight computational reduction (only slight, as we shall see that parallel transition resolving in the computation of the branch metrics becomes more difficult). One particularly troublesome feature of multidimensional trellis codes is, however, a propensity towards high nearest neighbor counts, with the resultant significant decrease in γ_f to $\tilde{\gamma}_f$.

Subsection B.5.2.1 introduces a few simple examples of multidimensional trellis codes. Subsection B.5.2.2 lists the most popular 4 dimensional codes in tabular form, similar to the tables in Section B.4.3. Subsection B.5.2.3 lists the most popular 8 dimensional codes.

B.5.2.1 Multidimensional Trellis Code Examples

EXAMPLE B.5.1 (2-state, rate 1/2, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 Z^4$. The two-state trellis is shown in Figure B.25. The labels for the various subsets are as indicated in the tables of Section B.5.1. The minimum distance is given by $d^2_{\text{min}} = 2$, and $\bar{r}_C = 0 + \frac{1}{4}$. Thus, the fundamental gain is

$$\gamma_f = \frac{2}{2^{2\cdot 1/4}} = \sqrt{2} \ \ (1.5 \text{ dB}) \quad . \tag{B.290}$$

This gain is no better than the D_4 gain, which required no states. However, $\bar{N}_e(D_4) = 6$, whereas for this code $\bar{N}_e = 2$, so the effective gain for the D_4 lattice code is about 1.2dB, while it is a full 1.5dB for this 2-state trellis code.

EXAMPLE B.5.2 (2-state, rate 1/2, 4D Code, based on D_4) The code uses $\Lambda = D_4$ and $\Lambda' = R_4 D_4$. The two-state trellis is shown in Figure B.26. The labels for the various subsets are as indicated on the tables in Section B.5.1. The minimum distance is given by

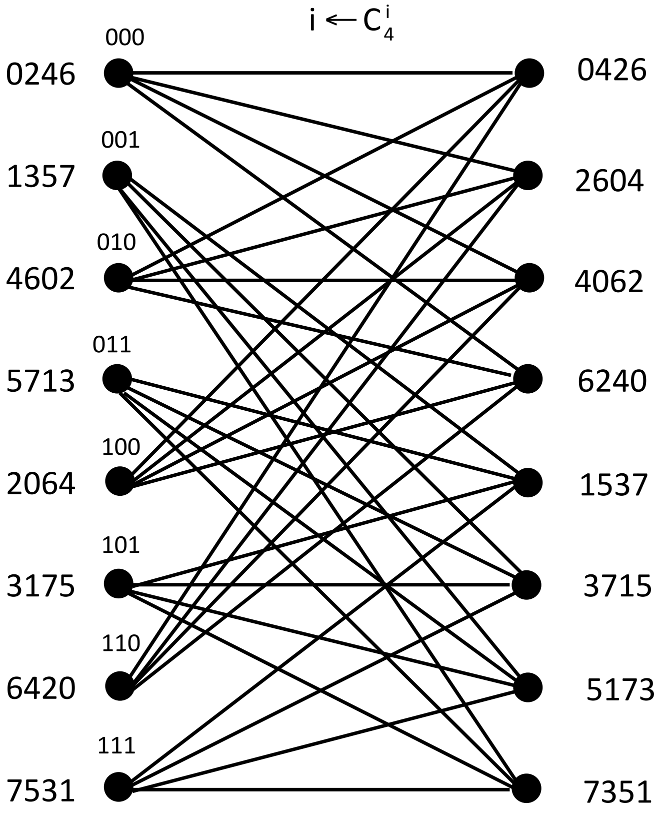


Figure B.27: Trellis for Wei's 8-state, rate 2/3, 4D Code (same as 8-state 2 dimensional Ungerboeck trellis, except branch index of i stands for coset C_4^i)

$d^2_{\min} = 4$, and $\bar{r_C} = \frac{1}{4} + \frac{1}{4}$.[4] Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/2}} = 2 \ \ (3.01 \text{ dB}) \quad . \tag{B.291}$$

This gain is better than the D_4 gain, which required no states. However, $\bar{N}_e = 22 \ \ (= \frac{24+8\times 8}{4})$, so the effective gain is about 2.31 dB for this code.

EXAMPLE B.5.3 (Wei's 8-state, rate 2/3, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 D_4$. The 8-state trellis is shown in Figure B.27. The labels for the various subsets are as indicated on the tables in Section B.5.1. The minimum distance is given by $d^2_{\min} = 4$, and $\bar{r_C} = 0 + \frac{1}{4}$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{B.292}$$

This gain is yet better. However, $\bar{N}_e = 22 \ \ (= \frac{24+8\times 8}{4})$, so the effective gain is about 3.83 dB for this code.

EXAMPLE B.5.4 (Wei's 16-state, rate 2/3, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 D_4$. The 16-state trellis is shown in Figure B.28. The minimum distance is given by

[4] Note that $\bar{r_\Lambda} = 1/4$ for $\Lambda = D_4$.

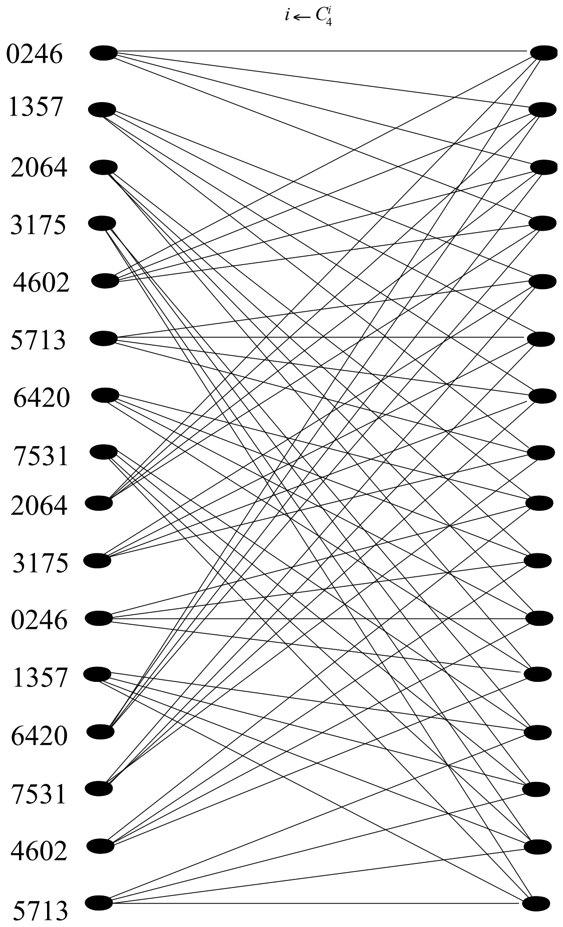


Figure B.28: Trellis for Wei's 16-state, rate 2/3, 4D code.

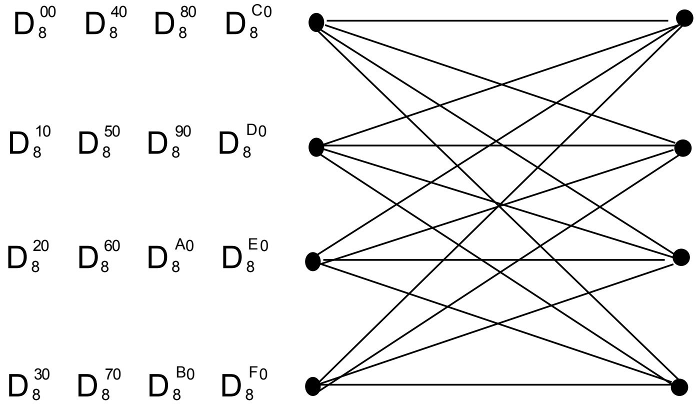

D_8^{00} D_8^{40} D_8^{80} D_8^{C0}
D_8^{10} D_8^{50} D_8^{90} D_8^{D0}
D_8^{20} D_8^{60} D_8^{A0} D_8^{E0}
D_8^{30} D_8^{70} D_8^{B0} D_8^{F0}

Figure B.29: Trellis for 4-state, rate 2/4, 8D Code

$d^2_{\text{min}} = 4$, and $\bar{r_C} = \frac{1}{4} + 0$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{B.293}$$

This gain is the same as for 8 states. However, $\bar{N}_e = 6$, so the effective gain is about 4.2 dB for this code, which is better than the 8-state code. This code is the most commonly found code in systems that do use a 4-dimensional trellis code and has been standardized as one option in the CCITT V.fast code for 28.8 Kbps (uncompressed) voiceband modems (with minor modification, see Section **??** and also for Asymmetric Digital Subscriber Line (ADSL) transceivers. There is a 32-state 4-dimensional Wei code with $\bar{N}_e = 2$, so that its effective gain is the full 4.52dB.

EXAMPLE B.5.5 (4-state, rate 2/4, 8D Code) The code uses $\Lambda = E_8$ and $\Lambda' = R_8 E_8$. The two-state trellis is shown in Figure B.29. The minimum distance is given by $d^2_{\text{min}} = 8$, and $\bar{r_C} = \frac{4}{8} + \frac{2}{8} = .75$.[5] Thus, the fundamental gain is

$$\gamma_f = \frac{8}{2^{2 \cdot 3/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{B.294}$$

This gain is better than for any other 4-state code presented so far. However, $\bar{N}_e = 126$ $(= \frac{240+3(16\times 16)}{8})$, so the effective gain is about 3.32 dB for this code, which is still better than the 4-state 2-dimensional 3 dB code with effective gain 3.01 dB.

EXAMPLE B.5.6 (Wei's 16-state, rate 3/4, 8D Code) The code uses $\Lambda = Z^8$ and $\Lambda' = E_8$. The 16-state trellis is shown in Figure B.30. The labels for the various subsets are as indicated on the tables in Section B.5.1. The minimum distance is given by $d^2_{\text{min}} = 4$, and $\bar{r_C} = \frac{1}{8} + 0 = .125$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/8}} = 2^{1.75} \ \ (5.27 \text{ dB}) \quad . \tag{B.295}$$

This gain is better than for any other 16-state code studied. However, $\bar{N}_e = 158$, so the effective gain is about 3.96 dB. There is a 64-state Wei code that is also $\gamma_f = 5.27$ dB, but with $\bar{N}_e = 30$, so that $\tilde{\gamma}_f = 4.49$ dB. This 64-state code (actually a differentially phase-invariant modification of it) is used in some commercial (private-line) 19.2kbps voiceband modems.

[5] Note that $\bar{r_\Lambda} = 4/8$ for $\Lambda = E_8$.

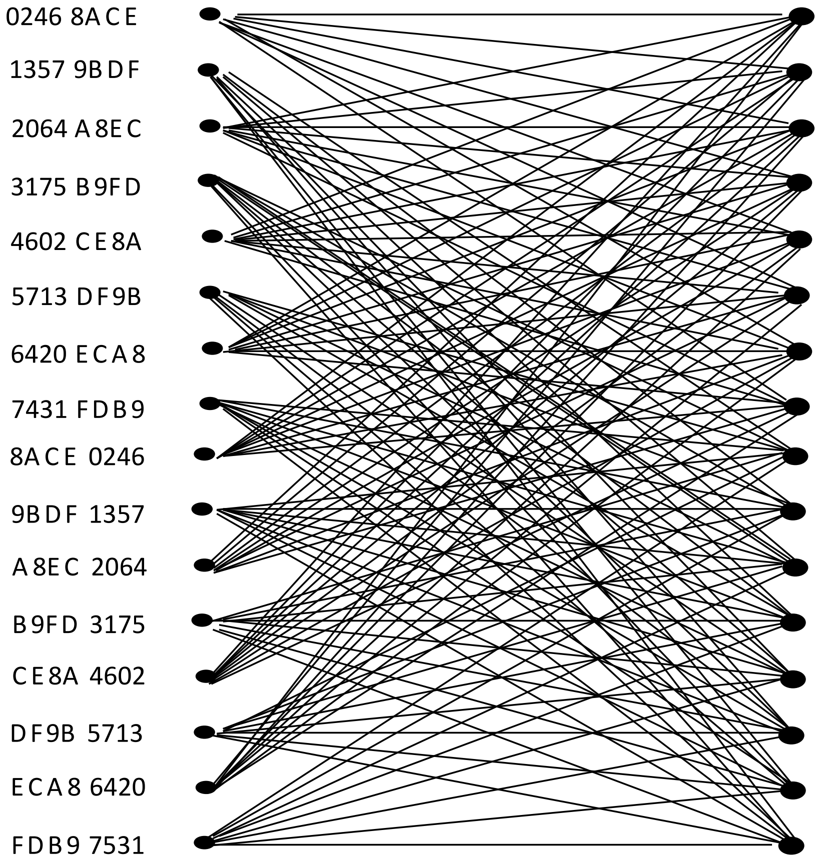


Figure B.30: Trellis for Wei's 16-state, rate 3/4, 8D code.

Λ	Λ'	2^ν	h_4	h_3	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\tilde{\gamma}_f$	$\bar{N}_D$
Z^4	R_4D_4	8	–	–	02	04	11	4	$2^{3/2}$	4.52	22	3.82	22
D_4	$2D_4$	16	–	10	04	02	21	6	3	4.77	88	3.88	76
Z^4	R_4D_4	16	–	–	14	02	21	4	$2^{3/2}$	4.52	6	4.20	36
Z^4	$2Z^4$	32	–	30	14	02	41	4	$2^{3/2}$	4.52	2	4.52	122
D_4	$2D_4$	64	–	050	014	002	121	6	3	4.77	8	4.37	256
Z^4	$2D_4$	64	050	030	014	002	101	5	$\frac{5}{\sqrt{2}}$	5.48	36	4.65	524
Z^4	$2D_4$	128	120	050	022	006	203	6	$\frac{6}{\sqrt{2}}$	6.28	364	4.77	1020

Table B.11: Four-Dimensional Trellis Codes and Parameters

Λ	Λ'	2^ν	h_4	h_3	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\tilde{\gamma}_f$	$\bar{N}_D$
E_8	R_8E_8	8	–	10	04	02	01	8	$2^{7/4}$	5.27	382	3.75	45
E_8	R_8E_8	16	–	10	04	02	21	8	$2^{7/4}$	5.27	158	4.01	60
Z^8	E_8	16	–	10	04	02	21	4	$2^{7/4}$	5.27	158	4.01	52
E_8	$2E_8$	16	r=4/8	-?-	-?-	-?-	-?-	16	4	6.02	510	4.42	342
E_8	R_8E_8	32	–	30	14	02	61	8	$2^{7/4}$	5.27	62	4.28	90
Z^8	E_8	32	–	10	04	02	41	4	$2^{7/4}$	5.27	62	4.28	82
E_8	R_8E_8	64	–	050	014	002	121	8	$2^{7/4}$	5.27	30	4.49	150
Z^8	E_8	64	–	050	014	002	121	4	$2^{7/4}$	5.27	30	4.49	142
Z^8	R_8D_8	128	120	044	014	002	101	8	$2^{7/4}$	5.27	14	4.71	516
E_8	$2E_8$	256	r=4/8	-?-	-?-	-?-	-?-	16	4	6.02	30	5.24	1272

Table B.12: Eight-Dimensional Trellis Codes and Parameters

B.5.2.2 4D Code Table

Table B.11 is similar to Tables B.1 and B.3, except that it is for 4 dimensional codes. Table B.11 lists up to rate 4/5 codes. The rate can be inferred from the number of nonzero terms h_i in the table. The design specifies Λ and Λ' in the 4D case, which also appears in Table B.11. $\bar{N}_1$ and $\bar{N}_2$ are not shown for these codes. $\bar{N}_1$ and $\bar{N}_2$ can be safely ignored because an increase in d_{min} in the tables by 1 or 2 would be very significant and the numbers of nearest neighbors would have to be very large on paths with $d > d_{\text{min}}$ in order for their performance to dominate the union bound for P_e.

B.5.2.3 8D Code Table

Table B.12 is similar to Tables B.1 and B.3, except that it is for 8 dimensional codes. Table B.12 lists up to rate 4/5 codes. The rate can again be inferred from the number of nonzero terms h_i in the table. Table B.12 specifies Λ and Λ' in the 8D case. $\bar{N}_1$ and $\bar{N}_2$ are also not shown for these codes, although the author suspects that $\bar{N}_1$ could dominate for the Z^8/R_8D_8 128-state code.

B.6 Theory of the Coset Code Implementation

This section investigates the simplification of the implementation of both the encoder and the decoder for a multidimensional trellis code.

B.6.1 Encoder Simplification

The general coset-code encoder diagram is useful mainly for illustrative purposes. Actual implementations using such a diagram as a guideline would require excessive memory for the implementation of the coset select and signal select functions. In practice, the use of two-dimensional constituent subsymbols in Z^2 is possible, even with codes that make use of more dense multidimensional lattices. All the lattices discussed in this Chapter and used are known as **binary lattices**, which means that they contain $2Z^N$ as a sublattice. With such binary lattices, additional partitioning can enlarge the number of cosets so that $2Z^N$ and its cosets partition the original lattice Z^N upon which the constellation was based. The number of additional binary partitions needed to get from Λ' to $2Z^N$ is known as the **informativity** of Λ', $k(\Lambda')$. This quantity is often normalized to the number of dimensions to get the **normalized informativity** of the lattice, $\bar{\kappa}(\Lambda) = k(\Lambda')/N$.

The addition of the extra partitioning bits to the coset code's convolutional encoder and coset selection creates a rate $[k + k(\Lambda')] \, / \, [k + k(\Lambda') + r_G]$ convolutional code that selects cosets of $2Z^N$. These cosets can then be trivially separated into two-dimensional cosets of $2Z^2$ (by undoing the concatenation). These 2D cosets can then can be independently specified by separate (reduced complexity and memory) two-dimensional signal selects. Wei's 16-state 4D code and 64-state 8D codes will illustrate this concept for multidimensional codes.

4D Encoder with rate 2/3, and Z^4/R_4D_4 An example is the encoder for a four-dimensional trellis code that transmits 10 bits per 4D symbol, or $b = 10$. The redundant extra bit from G ($r_G = 1$) requires a signal constellation with 2^{11} points in 4 dimensions. The encoder uses an uncoded constellation that is the Cartesian product of two 32CR constellations. 32CR is a subset of Z^2, and the uncoded constellation would be a subset of Z^4. A desirable implementation keeps the redundancy (extra levels) in the two 2D constituent sub-symbols as small as practically possible. The extension of 32CR to what is called 48CR appears in Figure B.31. 16 new points are added at the lowest 2D energy positions outside the 32CR to form 48CR. The 32CR points are denoted as "inner" points and the 16 new points are denoted as "outer" points. The Cartesian product of the two 48CR (a subset of Z^4) with deletion of those points that correspond to (outer, outer) form the coded constellation. This leaves 2^{10} (inner, inner) points, 2^9 (inner, outer) points, and 2^9 (outer, inner) points for a total of $2^{10} + 2 \cdot 2^9 = 2^{10} + 2^{10} = 2^{11} = 2048$ points. The probability of an inner point occuring is 3/4, while the probability of an outer is 1/4. The **two-dimensional** symbol energy is thus

$$\mathcal{E}_{\boldsymbol{x}}(two - dimensional) = \frac{3}{4} E_{32CR} + \frac{1}{4} E_{outer} = .75(5) + .25(\frac{2 \cdot 12.5 + 12.5 + 14.5}{4}) = 7 \quad , \tag{B.296}$$

so that $\bar{\mathcal{E}}_{\boldsymbol{x}} = 7/2$. The shaping gain for 48CR is then

$$\gamma_s = \frac{2^{2(1/4)} \cdot (2^5 - 1)}{12 \cdot (7/2)} = \frac{43.84}{42} = 1.0438 = .19 \; dB \quad . \tag{B.297}$$

The shaping gain of 32CR has already been computed as .14dB, thus the net gain in shaping is only about .05dB - however, a nominal deployment of a two-dimensional code would have required 64 points, leading to more constellation expansion, which might be undesirable. Since the code G is rate 2/3, the 3 output bits of G are used to select one of the 8 four-dimensional cosets, $C_{4,0}$, ..., $C_{4,7}$, and the 8 remaining bits would specify which of the parallel transitions in the selected $C_{4,i}$ would be transmitted, requiring a look-up table with 256 locations for each coset. Since there are 8 such cosets, the decoder would nominally need $8 \times 256 = 2048$ locations in a look-up table to implement the mapping from bits to transmitted signal. Each location in the table would contain 4 dimensions of a symbol. The large memory requirement can be mitigated to a large extent by using the structure illustrated in Figure B.32.

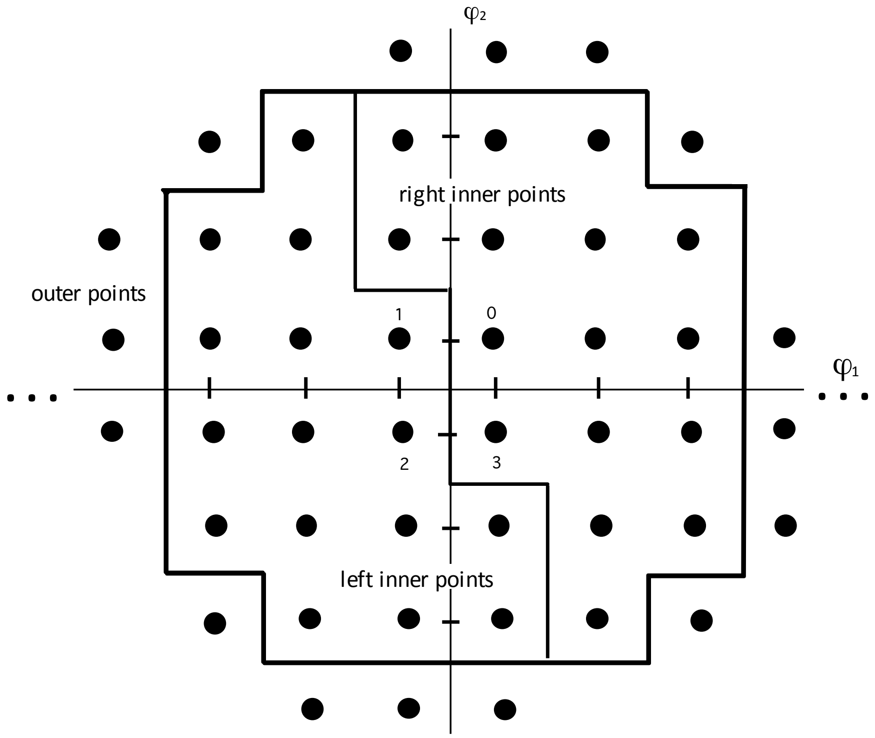


Figure B.31: 48 Cross constellation.

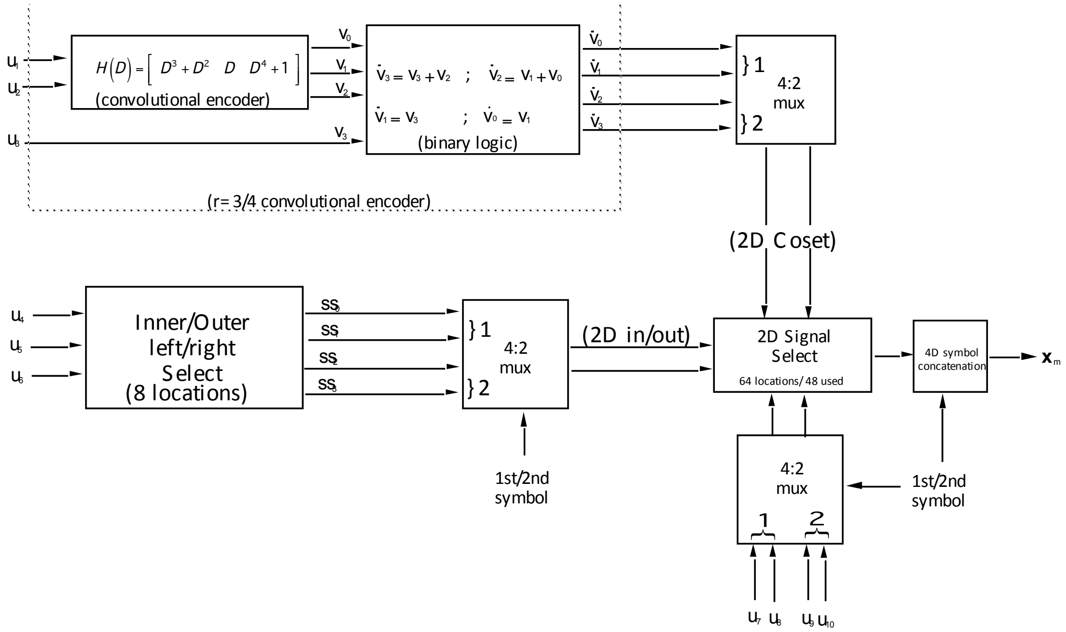


Figure B.32: 4D coset-select implementation.

Input Bits			1^{st} 2D subsym.			2^{nd} 2D subsym.		
u_4	u_5	u_6	ss_1	ss_0	position	ss_3	ss_2	position
0	0	0	0	0	left-inner	0	0	left-inner
0	0	1	0	0	left-inner	0	1	right-inner
0	1	0	0	0	left-inner	1	0	outer
0	1	1	0	1	right-inner	0	0	left-inner
1	0	0	0	1	right-inner	0	1	right-inner
1	0	1	0	1	right-inner	1	0	outer
1	1	0	1	0	outer	0	0	left-inner
1	1	1	1	0	outer	0	1	right-inner

Table B.13: Truth Table for 4D inner/outer selection (even b)

Because of the partitioning structure, each 4D coset of $C_{4,0}$ can be written as the union of two Cartesian products, for instance

$$C_4^0 = \left(C_2^0 \otimes C_2^0\right) \bigcup \left(C_2^2 \otimes C_2^2\right) \quad , \tag{B.298}$$

as in Section B.5. Bit v_3 specifies which of these two Cartesian products $\left(C_2^0 \otimes C_2^0\right)$ or $\left(C_2^2 \otimes C_2^2\right)$ contains the selected signal. These two Cartesian products are now in the desired form of cosets in $2Z^4$. Thus, $k(R_4 D^4) = 1$. The four bits v_0 ,..., v_3 can then be input into binary linear logic to form $\bar{v}_0$,..., $\bar{v}_3$. These four output bits then specify which 2D coset C_2^i, $i = 0, 1, 2, 3$ is used on each of the constituent 2D subsymbols that are concatenated to form the 4D code symbol. A clock of speed $2/T$ is used to control a 4:2 multiplexer that chooses bits $\bar{v}_0$ and $\bar{v}_1$ for the first 2D subsymbol and the bits $\bar{v}_2$ and $\bar{v}_3$ for the second 2D subsymbol within each symbol period. One can verify that the relations

$$\bar{v}_3 = v_2 + v_3 \tag{B.299}$$
$$\bar{v}_2 = v_0 + v_1 \tag{B.300}$$
$$\bar{v}_1 = v_3 \tag{B.301}$$
$$\bar{v}_0 = v_1 \tag{B.302}$$

will ensure that 4D coset, C_4^i, with i specified by $[v_2, v_1, v_0]$ is correctly translated into the two constituent 2D cosets that comprise that particular 4D coset.

The remaining input bits $(u_4, ..., u_{10})$ then specify points in each of the two subsymbols. The inner/outer/left/right select described in the Table B.13 takes advantage of the further separation of each 2D coset into 3 equal-sized groups, left-inner, right-inner, and outer as illustrated in Figure B.32. Note the left inner and right inner sets are chosen to ensure equal numbers of points from each of the 4 two-dimensional cosets. The outputs of this (nonlinear) bit map are then also separated into constituent 2D subsymbols by a multiplexer and the remaining 4 bits (u_7, u_8, u_9, u_{10}) are also so separated. The final encoder requires only 8 locations for the inner/outer selection and 64 locations (really only 48 are used because some of the combinations for the ss (signal select) bits do not occur) for the specification of each constituent subsymbol. This is a total of only 72 locations, significantly less than 2048, and only 2 dimensions of a symbol are stored in each location. The size of the 64-location 2D Signal Select is further reduced by Wei's observation that the 4 two-dimensional cosets can be generated by taking any one of the cosets, and performing sign permutations on the two dimensions within the subsymbol. Further, by storing an appropriate value (with signs) in a look-table, the necessary point can be generated by permuting signs according to the two bits being supplied for each 2D subsymbol from the CS function. Then the 2D signal selection would require only 16 locations, instead of 64. Then, the memory requirement would be (with a little extra logic to do the sign changes) 16+8=24 locations. We note $b \geq 6$ and b must be even for this encoder to work (while for $b < 6$ the encoder is trivial via look-up table with 2^{b+1} locations). For even $b > 6$, only the 2D Signal Select changes, and the memory size (using Wei's sign method to reduce by a factor of 4) is $4 \times 2^{\frac{b-6}{2}}$. For odd b, $b + 1$ is even, the constellation is square, and the inner/outer/left/right partitioning of the constituent 2D subsymbols is unnecessary.

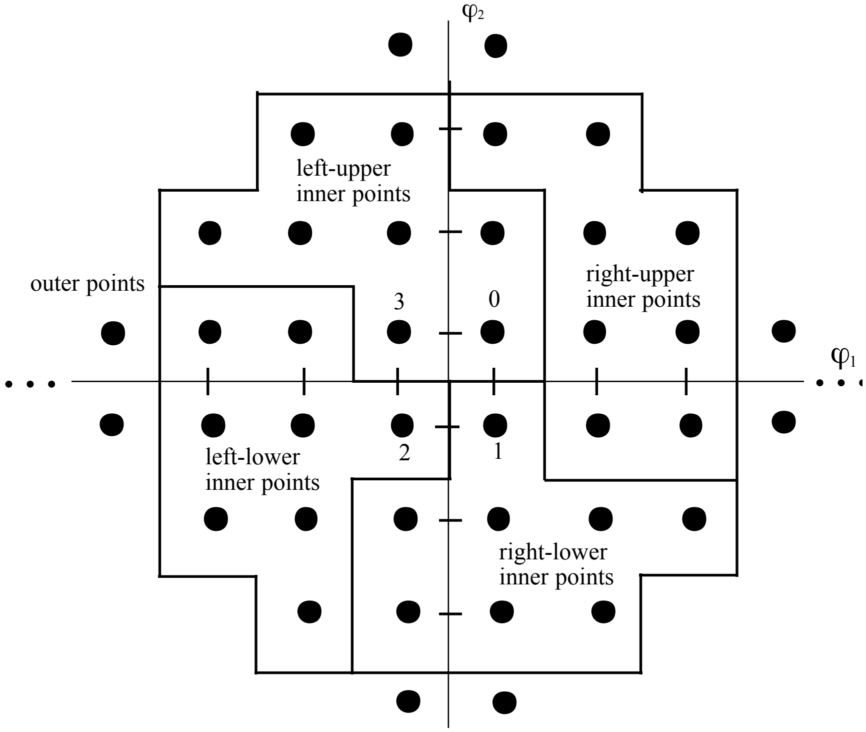


Figure B.33: 40 Cross constellation.

8D Encoder with rate 3/4, and Z^8/E_8 An encoder for $\bar{b} = 2.5$ and a 64-state eight-dimensional trellis code transmits 20 bits per 8D symbol, or $b = 20$. The redundant extra bit from G ($r_G = 1$) requires a signal constellation with 2^{21} points in 8 dimensions. The uncoded constellation is the Cartesian product of four 32CR constellations. Note that 32CR is a subset of Z^2, and our uncoded constellation would be a subset of Z^8. A desirable implementation keeps the redundancy (extra levels) in the two 2D constituent sub-symbols as small as is practically possible. 32CR extends to what is called 40CR and shown in Figure B.33. 8 new points are added at the lowest 2D energy positions outside the 32CR to form 40CR. The 32CR points are denoted as "inner" points and the 8 new points are denoted as "outer" points. The Cartesian product of the four 40CR (a subset of Z^8) with deletion of those points that have more than one outer point forms the 8-dimensional transmitted signal constellation. This leaves 2^{20} (in, in, in, in) points, 2^{18} (in, in, in, out) points, 2^{18} (in, in, out, in) points, 2^{18} (in, out, in, in) points, and 2^{18} (out, in, in, in) points for a total of $2^{20} + 4 \cdot 2^{18} = 2^{20} + 2^{20} = 2^{21} = 2,097,152$ points. The probability of an inner point occuring is 7/8, while the probability of an outer is 1/8. The **two-dimensional** symbol energy is thus

$$\mathcal{E}_{\boldsymbol{x}} = \frac{7}{8} E_{32CR} + \frac{1}{8} E_{outer} = .875(5) + .125(12.5) = 5.9375 \quad , \tag{B.303}$$

so that $\bar{\mathcal{E}}_{\boldsymbol{x}} = 5.9375/2$. The shaping gain for 40CR is then

$$\gamma_s = \frac{2^{2(1/8)} \cdot (2^5 - 1)}{12 \cdot (5.9375/2)} = \frac{36.8654}{35.625} = .15 \; dB \quad . \tag{B.304}$$

The shaping gain of 32CR has already been computed as .14dB, thus the net gain in shaping is only about .01dB - however, a nominal deployment of a two-dimensional code would have required 64 points, leading to more constellation expansion, which may be undesirable. Since the code G is rate 3/4, the 4 output bits of G are used to select one of the 16 eight-dimensional cosets, C_8^0, ..., C_8^F, and the 17 remaining bits would specify which of the parallel transitions in the selected C_8^i would be transmitted, requiring a look-up table with $2^{17} = 131,072$ 8-dimensional locations for each coset. Since there are 16 such cosets, the encoder would nominally need 2,097,152 8-dimensional locations in a look-up table to

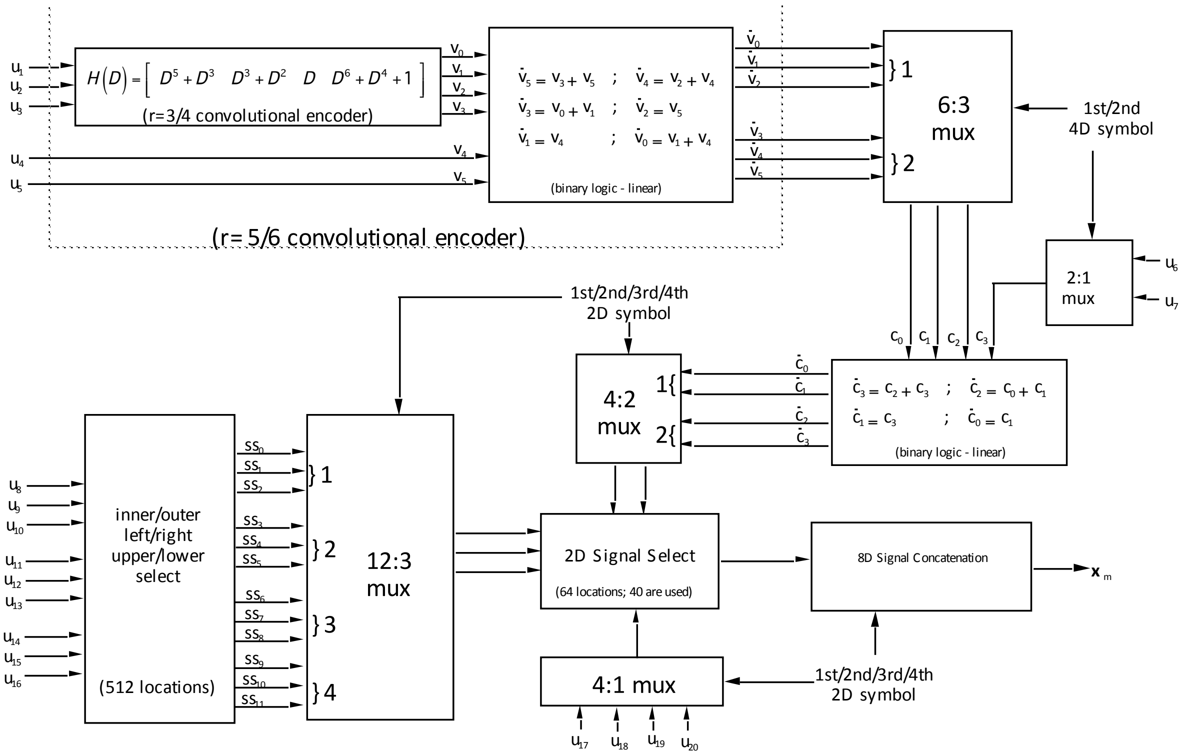


Figure B.34: 8D Coset Select Implementation

ss_2	ss_1	ss_0	position
0	0	0	left-top-inner
0	0	1	left-bottom-inner
0	1	0	right-top-inner
0	1	1	right-bottom-inner
1	0	0	outer

Table B.14: Truth Table for inner/outer selection (even b)

implement the mapping from bits to transmitted signal. The large memory requirement can be mitigated to a large extent by using the structure illustrated in Figure B.34. From the partitioning structure, each 8D coset of C_8^0 can be written as the union of four Cartesian products, for instance

$$C_8^0 = (C_4^0 \otimes C_4^0) \bigcup (C_4^2 \otimes C_4^2) \bigcup (C_4^4 \otimes C_4^4) \bigcup (C_4^6 \otimes C_4^6) \quad , \tag{B.305}$$

and each of these Cartesian products is a $R_4 D_4$ sublattice coset. Bits u_4 and u_5 specify which of these four Cartesian products contains the selected signal. The original convolutional encoder was rate 3/4, and it now becomes rate 5/6. The six bits v_0 ... v_5 can then be input into a linear circuit with the relations illustrated in Figure B.34 that outputs six new bits in two groups of three bits each. The first such group selects one of the eight 4D cosets C_4^0 ... C_4^7 for the first 4D constituent sub-symbol and the second group does exactly the same thing for the second constituent 4D subsymbol. The encoder uses two more bits u_6 and u_7 as inputs to exactly the same type of linear circuit that was used for the previous four-dimensional code example. We could have redrawn this figure to include u_6 and u_7 in the convolutional encoder, so that the overall convolutional encoder would have been rate 7/8. Note that $k(E_8) = 4$ (u_4, u_5, u_6, and u_7). The nine bits u_8 ... u_{16} then are used to specify inner/outer selection abbreviated by Table B.14. The 9 output bits of this inner/outer selection follow identical patterns for the other 2D constituent subsymbols. There are 512 combinations ($4^4 + 4 \cdot 4^3$), prohibiting more than one outer from occuring in any 8D symbol.

The final encoder requires only 512 locations for the "inner/outer/left/right/upper/lower" circuit, and 64 locations (only 40 actually used) for the specification of each constituent subsymbol. This is a total of only 552 locations, significantly less than 2,097,152, and each is only two-dimensional. Further reduction of the size of the 64-location memory occurs by noting that the 4 two-dimensional cosets can be generated by taking any one of the cosets, and performing sign permutations on the two dimensions within the subsymbol. Then, the memory requirement would be (with a little extra logic to do the sign changes) 528 locations.

B.6.2 Decoder Complexity

The straightforward implementation of this maximum likelihood sequence detector for 4D and 8D codes can be very complex. This is because of the (usually) large number of parallel transitions between any two pairs of states. In one or two dimensions with (possibly scaled) Z or Z^2 lattices, the closest point to a received signal is found by simple truncation. However, the determination of the closest point within a set of parallel transitions that fall on a dense 4D or 8D lattice can be more difficult. This subsection studies such decoding for both the D_4 and E_8 lattices. A special form of the Viterbi Algorithm can also readily be used in resolving the closest point within a coset, just as another form of the Viterbi Algorithm is useful in deciding which sequence of multidimensional cosets is closest to the received sequence.

Decoding the D_4 Lattice By definition:

$$D_4 = R_4Z^4 \bigcup \left(R_4Z^4 + [0,1,0,1]\right) \tag{B.306}$$

$$= \left(R_2Z^2 \otimes R_2Z^2\right) \bigcup \left((R_2Z^2 + [0,1]) \otimes (R_2Z^2 + [0,1])\right) \quad , \tag{B.307}$$

which can be illustrated by the trellis in Figure B.18. Either of the two paths through the trellis describes a sequence of two 2D subsymbols which can be concatenated to produce a valid 4D symbol in D_4. Similarly, upon receipt of a 4D channel output, the decoder determines which of the paths through the trellis in Figure B.18 was closest. This point decoder can use the Viterbi Algorithm to perform this decoding function. In so doing, the following computations are performed:

trellis position	subsymbol (1 or 2)	adds (or compares)
R_2Z^2	1	1 add
R_2Z^2	2	1 add
$R_2Z^2 + [0,1]$	1	1 add
$R_2Z^2 + [0,1]$	2	1 add
middle states	–	2 adds
final state	–	1 compare
Total	1 & 2	7 ops

The R_4D_4 Lattice has essentially the same lattice as shown in Figure B.35. The complexity for decoding R_4D_4 is also the same as in the D_4 lattice (7 operations).

To choose a point in each of the 8 cosets of $\Lambda' = R_4D_4$ in a straightforward manner, the Viterbi decoding repeats 8 times for each of the 8 cosets of R_4D_4. This requires 56 operations for the coset determination alone. However, 32 operations are sufficient: First, the two cosets of D_4 partition Z_4 into two parts, as illustrated in Figure B.36. The 2 dimensional cosets $B_{2,0}$ and $B_{2,1}$ for both the first 2D subsymbol and the second 2D subsymbol are common to both trellises. Once the decoder has selected the closest point in either of these two cosets (for both first subsymbol and again for the second subsymbol), it need not repeat that computation for the other trellis. Thus, the decoder needs 7 computations to decode one of the cosets, but only 3 additional computations (2 adds and 1 compare) to also decode the second coset, leaving a total of 10 computations (which is less than the 14 that would have been required had the redundancy in the trellis descriptions for the two cosets not been exploited).

Returning to R_4D_4, the 8 trellises describing the 8 cosets, C_4^i, $i = 0, ..., 7$, used in a 4D (Z^4/R_4D_4) trellis code are comprised of 4 2D cosets (C_2^0, C_2^1, C_2^2 and C_2^3) for both the first 2D subsymbol and for the second 2D subsymbol. The decoding of these cosets (for both subsymbols) thus requires 8 additions. Then the decoding performs 3 computations (2 adds and 1 compare) for each of the 8 trellises corresponding to the 8 cosets, requiring an additional 24 computations. Thus, the total computation

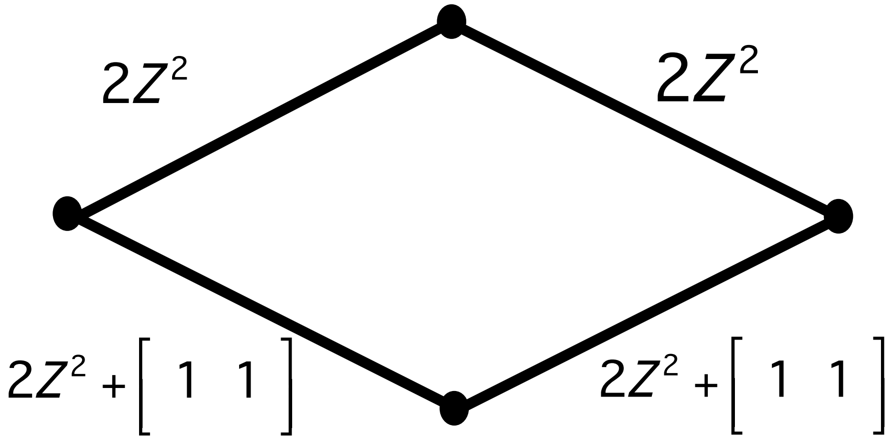


Figure B.35: Trellis for the R_4D_4 Lattice

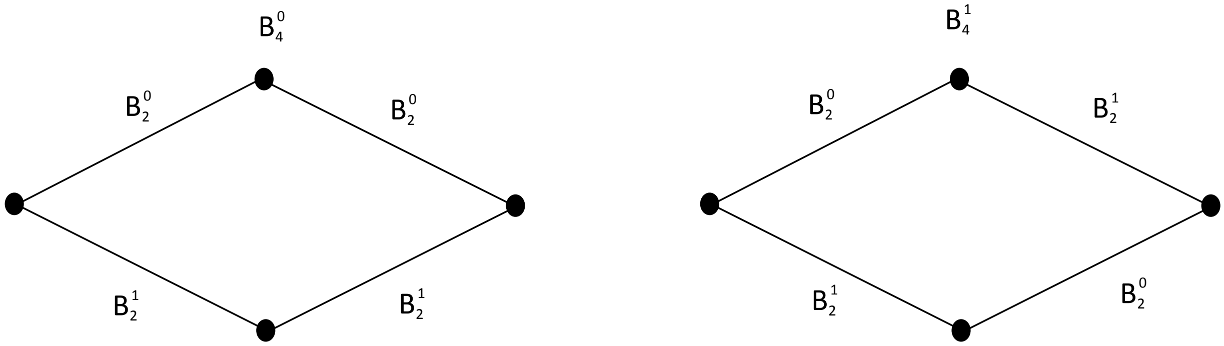


Figure B.36: Trellis for the two cosets of the D_4 lattice

required to decode all 8 cosets is 32 computations. Normalized to 1 dimension, there are 8 computations for all 8 cosets of the D_4 lattice.

We can now return to the complexity of decoding the entire coset code. The complexity of decoding using the Viterbi detector for the trellis code, after parallel transitions have been resolved is

$$N_D(trellis) = 2^{\nu}\left(2^k \text{ adds } + 2^k - 1 \text{ compares}\right) \quad . \tag{B.308}$$

The overall complexity $N_D(C)$ is the sum of the complexity of decoding the 2^{k+r_G} cosets of Λ' plus the complexity of decoding the sequences of multidimensional codewords produced by the trellis code.

> **EXAMPLE B.6.1 (Wei's 16-state 4D code, with $r = 2/3$)** This code is based on the partition Z^4/R_4D_4 and thus requires 32 operations to decode the 8 sets of parallel transitions. Also as $k = 2$ and $2^{\nu} = 16$, there are $16(4{+}3) = 112$ operations for the remaining 4D sequence detection. The total complexity is thus $112 + 32 = 144$ operations per 4D symbol. Thus $\bar{N}_D = 144/4 = 36$, which was the corresponding entry in Table B.11 earlier.

The rest of the $\bar{N}_D$ entries in the 4D table are computed in a similar fashion.

B.6.3 Decoding the Gossett (E_8) Lattice

The decoding of an E_8 lattice is similar to the decoding of the D_4 lattice, except that it requires more computation. It is advantageous to describe the decoding of the 16 cosets of E_8 in Z^8 in terms of the easily decoded 2D constituent symbols.

The E_8 lattice has decomposition:

$$\begin{aligned} E_8 &= R_8D_8 \bigcup (R_8D_8 + [0,1,0,1,0,1,0,1]) & \text{(B.309)} \\ &= (R_4D_4)^2 \bigcup (R_4D_4 + [0,0,1,-1])^2 & \text{(B.310)} \\ &\quad \bigcup (R_4D_4 + [0,1,0,1])^2 \bigcup (R_4D_4 + [0,1,1,0])^2 \quad , & \text{(B.311)} \end{aligned}$$

which uses the result

$$R_8D_8 = (R_4D_4)^2 \bigcup (R_4D_4 + [0,0,1,-1])^2 \quad . \tag{B.312}$$

The trellis decomposition of the E_8 in terms of 4D subsymbols as described by (B.311) is illustrated in Figure B.20. The recognition that the 4D subsymbols in Figure B.20 can be further decomposed as in Figure B.18 leads to Figure B.21. The coset leader $[1,-1]$ is equivalent to $[1,1]$ for the "C-level" cosets in two dimensions, which has been used in Figures B.20 and B.21.

The complexity of decoding the E_8 lattice requires an addition for each of the 4 cosets of R_2D_2 for each of the 4 2D subsymbols, leading to 16 additions. Then for each of the (used) 4 cosets of R_4D_4, decoding requires an additional (2 adds and 1 compare) leading to 12 computations for each 4D subsymbol, or 24 total. Finally the 4D trellis in Figure B.20 requires 4 adds and 3 compares. The total is then 16+24+7=47 operations.

All 16 cosets of the E_8 lattice require 16 additions for the 4 2D C-level partitions, and 48 computations for both sets (first 4D plus second 4D) of 8 cosets of R_4D_4 that are used, and finally 16(7)=112 computations for decoding all sixteen versions of Figure B.20. This gives a total of $N_D = 16 + 48 + 112 = 176$, or $\bar{N}_D = 22$.

> **EXAMPLE B.6.2 (Wei's 64-state 8D code, with $r = 3/4$)** This code is based on the partition Z^8/E_8 and requires 176 operations to decode the 16 cosets of E_8. Also with $k = 3$ and $2^{\nu} = 64$, there are $64(8 + 7) = 960$ operations for the remaining 8D sequence detection. The total complexity is thus $960 + 176 = 1136$ operations per 8D symbol. Thus $\bar{N}_D = 1136/8 = 142$, which was the corresponding entry in Table B.12 earlier.

B.6.4 Lattice Decoding Table

Table B.15 is a list of the decoding complexity to find all the cosets of Λ' in Λ.

Λ	Λ'	$\lvert\Lambda/\Lambda'\rvert$	N_D	$\bar{N}_D$
Z^2	D_2	2	2	1
Z^2	$2Z^2$	4	4	2
Z^2	$2D_2$	8	8	4
Z^2	$4Z^2$	16	16	8
Z^4	D_4	2	10	2.5
Z^4	R_4D_4	8	32	8
Z^4	$2D_4$	32	112	28
Z^4	$2R_4D_4$	128	416	104
D_4	R_4D_4	4	20	5
D_4	$2D_4$	16	64	16
D_4	$2R_4D_4$	64	224	56
Z^8	D_8	2	26	3.25
Z^8	E_8	16	176	22
Z^8	R_8E_8	256	2016	257
Z^8	$2E_8$	4096	29504	3688
D_8	E_8	8	120	15
D_8	R_8E_8	128	1120	140
D_8	$2E_8$	2048	15168	1896
E_8	R_8E_8	16	240	30
E_8	$2E_8$	256	2240	280

Table B.15: Lattice Decoding Complexity for all Cosets of Selected Partitions

B.7 Various Results in Encoder Realization Theory

Convolutional codes can have many implementations, which correspond to different encoders that map the same set of codewords with different input sequences. There is a rich theory that permits any code's realization with minimum number of delays (so minimizing decoder/trellis number of states), with feedback-free encoder and decoder, or with systematic often-feedback-based encoder. The theory in particular helps map $r = n/(n+1)$ codes specified more efficiently by $H(D)$ into minimal realizations. The original theory is a masterpiece largely pioneered by G. David Forney at the coding world's genesis. The theory is not necessary for rate $r = 1/2$ or really any $r = 1/n$ codes as they trivially minimize easily. Modern codes often instead puncture $r = 1/2$ base codes to retain constant gap, and retaining the trivial realization. This appendix, however, preserves and provides the theory for potential use. One advantage is this theory allows circumvention of many publicly available decoder programs (for instance matlab's vitdec.m program with associated poly2trellis.m that exponentially increase the number of states in their use with respect to the minimum necessary to decode. This appendix section provides a way to "spoof" these programs into doing optimal decoding with a minimal number of states.

Subsection B.7.1 begins by providing the **Invariant Factors Decomposition (IFD)** of a finite-field transfer function. This is the finite-field (and this Appendix really only handles binary convolutional codes, but the extensions are self evident for the interested reader) equivalent of Smith Canonical Factorization over a real (or complex) transfer function, the latter sometimes used by control theorists. IFD creates a foundation for later subsections that provide methods to derive minimal feedback-free and systematic-with-feedback realizations. The Smith-Canonical form also follows for those interested by simply replacing the finite field $GF2$ with the field $\mathbb{R}$ or $\mathbb{C}$ (and perhaps the placeholder variable D with the Laplace Transform variable s). Section B.7.2 covers the use of IFD to construct equivalent basic and minimal encoders, which have minimum number of states and are feedback free, with feedback-free inverses. Systematic realizations with minimal states, but possibly using feedback, also appear.

B.7.1 Invariant Factors Decomposition

This section presents the IFD algorithm that factors any encoder generator matrix $G(D)$. This text has immediate interest only sequences $FD]$, which as in Section 8.1 is the ring of polynomial sequences with coefficients in $GF2$. The process extends also later to sequences in the field $F_r(D)$, which allows for denominator polynomials (and thus feedback or "IIR" filters that describe binary sequencies).

> **Definition B.7.1 [Invariant Factors Decomposition]** *The IFD of a $k \times n$ generator matrix $G(D) \in \{F[D]\}^{k\times n}$ is*
>
> $$G(D) = A(D) \cdot \Gamma(D) \cdot B(D) \quad , \tag{B.313}$$
>
> *where the $k \times k$ matrix $A(D) \in \{F[D]\}^{k\times k}$ is unimodulator so that $|A(D)| = 1$, as is similarly the $n \times n$ matrix $B(D) \in \{F[D]\}^{n\times n}$ with $|B(D)| = 1$. $\Gamma(D)$ is a diagonal matrix of invariant factors $\gamma_i(D) \in F[D]$ along the diagonal.*

The matrices $A(D)$, and $B(D)$, conform to a series of operations that replace any row, or column, with a sum of it the present value and any multiple (in $F[D]$) of another row (column), as well as row and/or column interchanges.

The IFD Algorithm:

1. Extract the least-common multiple, $\phi(D)$ of any denominator terms by forming $\tilde{G}(D) = \phi(D) \cdot G(D)$, thus making the elements of $G(D)$ polynomials in $f[D]$.

2. Implement row and/or column switches to move the nonzero element with lowest degree (in D) to the (1,1) position, and *keep track of the operations.*

3. Use the (1,1) entry together with row and column operations to eliminate (if possible) all other elements in the first row and column. Use only polynomials in D; you may not be able to eliminate

the element, but still try to eliminate as much of it (in terms of D^k quantities) as possible. *Keep track of the operations here also.*

An example that places "1" in the upper position with a row operation is

$$\left[\begin{array}{c} D \\ (1+D^3) \end{array} \right. \cdots \begin{array}{l} (D^2) \times \text{row } 1 + \text{row } 2 = D^3 + (1+D^3) = 1 \\ \text{in (1,1) spot. – can't do better.} \end{array} \tag{B.314}$$

4. If (1,1) entry is now not of the lowest degree in D, the IFD Algorithm return to step 2.

5. With the (1,1) entry now of lowest degree, the IFD algorithm repeats step 2 except that it now moves *next lowest* degree term in D to (2,2). Then, the IVP repeats 3.

This (tedious) process, produces the matrix of invariants, Γ. It is the same procedure as is used to find the Smith-Macmillan form of a transfer matrix for a MIMO linear system, however the author is unaware of any present matlab (or other) software that implements this procedure in $GF2$ (The matlab Smith Canonical software typically runs on real or complex coefficient transforms.). This section's notational simplification further drops the (D) on A, B, and Γ, but they each are implied functions of D with elements in $F[D]$. N The row and column operations create the A matrix' inverse through some finite number, N, of successive left-side $k \times k$ matrix multiplications $L_i, i = 1, ..., N$ and another number, M, of successive right-side $n \times n$ matrix multiplications $R_j, j = 1, ..., M$. The elements of L_i and R_j also are in $f[D]$.

$$L_N \cdots L_2 \cdot L_1 \cdot G \cdot R_1 \cdot R_2 \cdots R_M = \Gamma \ \ . \tag{B.315}$$

Therefore,

$$G = (L_N \cdots L_2 \cdot L_1)^{-1} \cdot \Gamma \cdot (R_1 \cdot R_2 \cdots R_M)^{-1} = A \cdot \Gamma \cdot B \tag{B.316}$$

where $A = (L_N \cdots L_2 \cdot L_1)^{-1} = L_1^{-1} L_2^{-1} \cdots L_N^{-1}$ and $B = (R_1 R_2 \cdots R_j)^{-1} = R_j^{-1} R_{j-1}^{-1} \cdots R_1^{-1}$.

IFD Observations: Some IFD observations include:

1. $(L_i \cdots L_2 \cdot L_1)^{-1}$ and $(R_1 \cdot R_2 \cdots R_j)^{-1}$ are unimodular because each L_i and R_j is individually unimodular, with determinant $|L_i| = |R_j| = 1$; so the determinants of the products $(\cdots L_2 \cdot L_1)$ and $(\cdots R_1 \cdot R_2)$ are also 1; consequently, the matrices A and B's inverses exist and all elements remain polynomials in $F[D]$ via step 3's allowed operations.

2. The IFD's construction is thus $G = A \cdot \Gamma \cdot B = (L_1^{-1} \cdots L_i^{-1})\Gamma(R_j^{-1} \cdots R_1^{-1})$. A and B are not unique, but Γ is unique for the given generator G.

3. Matrix inversions when elements are binary-coefficient polynomials are easy - just calculate the determinants of the adjoints, since the denominator determinant is 1 (using Appendix C's Cramer Rule for inverse construction).

EXAMPLE B.7.1 **[8-state rate 3/4 code]** A rate $r = 3/4$ code has systematic generator

$$G(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} = \frac{1}{1+D^3} \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D^2 \\ 0 & 0 & 1+D^3 & D \end{bmatrix} \ \ . \tag{B.317}$$

(B.317)'s encoder $G(D)$ immediately violates the $f[D]$ elements constraint. Instead the designer finds the IFD of

$$\tilde{G}(D) = \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D^2 \\ 0 & 0 & 1+D^3 & D \end{bmatrix} \ , \tag{B.318}$$

where eventually the $1+D^3$ factor may be divided from IFD results to return feedback to the implementation, without code change. The term of lowest degree is D, so the IFD Algorithm forms[6] L_1 to swap rows 1 and 3 and also R_1 to swaps columns 1 and 4: i.e.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \tilde{G}(D) \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = L_1 \cdot \tilde{G} \cdot R_1 = \begin{bmatrix} D & 0 & 1+D^3 & 0 \\ D^2 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix}$$

The IFD Algorithm next eliminates the remaining entries in column 1 by using ⇒ row 2 → row 2 + D row 1 or

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ D & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{L_2} \cdot \underbrace{\begin{bmatrix} D & 0 & 1+D^3 & 0 \\ D^2 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix}}_{L_1 \cdot \tilde{G} \cdot R_1} = \underbrace{\begin{bmatrix} D & 0 & 1+D^3 & 0 \\ 0 & 1+D^3 & D(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix}}_{L_2 \cdot L_1 \cdot \tilde{G} \cdot R_1} .$$

Now eliminate entries in row 1, by ⇒ column 3 → column 3 + D^2 (not $\frac{D}{1+D^3}$, as we must have a $f[D]$ matrix) × column 1

$$= \underbrace{\begin{bmatrix} D & 0 & 1+D^3 & 0 \\ 0 & 1+D^3 & D(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix}}_{L_2 \cdot L_1 \cdot \tilde{G} \cdot R_1} \cdot \underbrace{\begin{bmatrix} 1 & 0 & D^2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R_2}$$

$$= \underbrace{\begin{bmatrix} D & 0 & 1 & 0 \\ 0 & 1+D^3 & D(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix}}_{L_2 \cdot L_1 \cdot \tilde{G} \cdot R_1 \cdot R_2}$$

IFD Algorithm then moves the (1,3) element to (1,1), since it is now the nonzero element with lowest degree in D, so column 1 ↔ column 3:

$$L_2 \cdot L_1 \cdot \tilde{G} \cdot R_1 \cdot R_2 \cdot \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R_3} = \begin{bmatrix} 1 & 0 & D & 0 \\ D(1+D^3) & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \quad \text{(B.320)}$$

⇒ row 2 → row 2 + D(1+D^3) × row 1 and ⇒ column 3 → column 1 $\cdot D$ + column 3 yields

$$\tilde{G}' = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ D(1+D^3) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{L_3} \cdot \begin{bmatrix} 1 & 0 & D & 0 \\ D(1+D^3) & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 0 & D & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R_4}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & D^2(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \quad \text{(B.321)}$$

[6] Note that

$$L_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \text{ swaps rows 1 \& 3 and } R_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \text{ swaps columns 1 \& 4} \quad \text{(B.319)}$$

This completes (1,1) term portion. Continuing IFD Algorithm adds D^2 times column 2 to column 3 to complete the (2,2) element and then finally swaps columns 3 and 4.

$$\begin{aligned}\tilde{G}'' &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & D^2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R_5} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{R_6} \\ &= \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 1+D^3 & 0 \end{bmatrix}}_{\Gamma} . \end{aligned} \tag{B.322}$$

Therefore

$$\Gamma = L_3 \cdot L_2 \cdot L_1 \cdot \tilde{G} \cdot R_1 \cdot R_2 \cdot R_3 \cdot R_4 \cdot R_5 \cdot R_6 \tag{B.323}$$

with

$$L_3 \cdot L_2 \cdot L_1 = L = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & D^4 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{and} \tag{B.324}$$

So

$$A = L^{-1} = \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{B.325}$$

Similarly

$$R_1 \cdot R_2 \cdot R_3 \cdot R_4 \cdot R_5 \cdot R_6 = R \tag{B.326}$$

and

$$B = R^{-1} = \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} \tag{B.327}$$

$$\tilde{G} = A \cdot \Gamma \cdot B = \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 1+D^3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} \tag{B.328}$$

Restoring the $\frac{1}{1+D^3}$ factor,

$$G = \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} = A\Gamma B \tag{B.329}$$

EXAMPLE B.7.2 **[Rate 2/3 code with no feedback]**

$$G(D) = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \tag{B.330}$$

The IFD Algorithm executes col 2 ⇒ col 2 + D × col 1

$$G(D) \cdot \underbrace{\begin{bmatrix} 1 & D & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_1} = \begin{bmatrix} 1 & 0 & 0 \\ D^2 & 1+D^3 & D \end{bmatrix} \tag{B.331}$$

Then, row 2 ⇒ row 2 + D^2 × row 1

$$\underbrace{\begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix}}_{L_1} \cdot \begin{bmatrix} 1 & 0 & 0 \\ D^2 & 1+D^3 & D \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & D \end{bmatrix} \tag{B.332}$$

col 2 ⇒ col 3

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & D \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{R_2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1+D^3 \end{bmatrix} \tag{B.333}$$

col 3 ⇒ col 3 + D^2 × col 2

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1+D^3 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D^2 \\ 0 & 0 & 1 \end{bmatrix}}_{R_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1 \end{bmatrix} \tag{B.334}$$

col 2 ⇒ col 3 and col 3 ⇒ col 3 + D × col 2 provides

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{R_4} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{bmatrix}}_{R_5} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{\Gamma}$$

$$L = L_1 = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix} \text{ so } A = L_1^{-1} = L = L_1 = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix}$$

$$R = R_1 \cdot R_2 \cdot R_3 \cdot R_4 \cdot R_5 = \begin{bmatrix} 1 & D & D^3 \\ 0 & 1 & D \\ 0 & D^2 & 1+D^3 \end{bmatrix} \text{ and } B = R^{-1} = \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \\ 0 & D^2 & 1 \end{bmatrix}$$

Therefore

$$G = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \\ 0 & D^2 & 1 \end{bmatrix} \tag{B.335}$$

B.7.2 Canonical Realizations of Convolutional Encoders

Subsection B.7.1 earlier discussed equivalent encoders, for which the (output) codes were the same. Any code can be realized by canonical encoders that have several useful properties:

1. $G(D)$ has a feedback-free right inverse, $G^{-1}(D)$, an $n \times k$ matrix, such that $G(D)G^{-1}(D) = I$, that is, all entries of $G^{-1}(D)$ are in $F[D]$. This can be helpful if decoded sequences for another encoder realization of the same code are available, So the receiver could re-encode them with the old encoder, invert the output with the feedback-free inverse, and re-encode with the same-code, but differernt mapping encoder, This is then MLSD result for the other new encoder without need of another decoder. This can be helpful circumventing a limitation where a decoder was designed for another encoder of the same code.

2. $G(D)$ is feedback free ($G(D)$ also has all entries in $F[D]$).

3. ν is a minimum, that is $\nu = \mu$.

This section reviews such encoders and provides the translation process of a non-canonical encoder into canonical form. An encoder matrix $G(D)$'s IFD is central to its reduction to canonical form. Subsection B.7.2.1 formalizes the existence of and provides further IFD examples.. The minimal encoder's realization uses a minimum number of delay elements. A minimal encoder will need to be delay preserving (i.e. input sequences are not delayed), degree preserving (i.e., an interesting dual property to delay preservation for sequences running backwards in time), and non-catastrophic – Subsection B.7.2.4 studies these properties and proves the equivalence of these properties to a minimal encoder. The minimal encoder's construction is nontrivial and requires a two step procedure: The first step constructs the basic as in Subsection B.7.2.5. The second step produces the minimal encoder from a basic encoder, as in Section B.7.2.6. Systematic encoders can also be found with the possible need of feedback. Subsection B.7.2.7 provides the canonical systematic encoder with feedback.

B.7.2.1 Invariant Factors

A polynomial matrix $G(D)$'s IFD is the finite-field version of the Smith Canonical Matrix form of basic mathematics and signal processing. Here, IFD is for matrices $G(D) \in \{F[D]\}^{k\times n}$. It applies to nonsquare matrices and provices the factorization $A(D)\cdot\Gamma(D)\cdot B(D)$ where A and B have unit determinants (unimodular). This differs from singular value decomposition where the F and M matrices, which initially appear ivery similar to A and B, are unitary (a stronger restriction that simple unit determinant). IFD's diagonal **invariant factors** can be found in polynomial time with an exact algorithm (unlike SVD). They represent the ratios of successive greatest common divisors of the original matrix' submatrix determinants, while the singular values require an iterative approximation algorithm to compute and represent components of the matrix' action on two linked orthonormal bases in a transformation from a n-dimensional space to a k-dimensional space. The invariant factors themselves are less important in coding than the resultant upper k rows of B that find a certain basic encoder with desirable properties.

While $G(D) \in \{F[D]\}^{k\times n}$, any $G(D)$ with entries in $F_r(D)$ transforms to another feedback-free encoder by premultiplying the original generator by $A = \phi(D)\cdot I$ ($A \in F(D)$), where $\phi(D)$ is the least common multiple of all the original generator's feedback (denominator) polynomials. This premultiplication does not change the codewords generated by $G(D)$. Again, notational simplification drops the D from developments. Formally:

Theorem B.7.1 **[Invariant Factors Theorem]** *Let G be a $k\times n$ matrix of polynomials in $F[D]$ of rank $\varrho_G = k$, then*

$$G = A\Gamma B \tag{B.336}$$

where

1. A and B are square $k\times k$ and $n\times n$ $F[D]$ matrices with unit determinants $|A| = 1$ and $|B| = 1$, respectively, that is they are **unimodular**.
2. A^{-1} and B^{-1} are also square $F[D]$ matrices.
3. Γ is a $k\times n$ matrix with the following structure:

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_k & 0 & \dots & 0 \end{bmatrix} \quad . \tag{B.337}$$

4. $\gamma_i \in f[D]$ are the unique **invariant factors** for $G(D)$ and

$$\gamma_i = \frac{\Delta_i}{\Delta_{i-1}} \quad . \tag{B.338}$$

5. Δ_i is the greatest common divisor (GCD) of all determinants of $i\times i$ submatrices of $G(D)$, with $\Delta_0 = 1$.

6. γ_i divides γ_{i+1}, that is γ_i is a factor of γ_{i+1}.

Proof: As in Subsection B.7.2.1's examples, any unimodular A (B) comprise a product of matrices that are equivalent to elementary row (column) operations of adding a multiple (in $F[D]$) of one row (column) to another row (column). If G is not already diagonal, let α and β be elements in the same column where α does not divide β (if it always did, there would have to be a β in some other column that could be added to the present column to make β not divisible by α, or the dimensionality would be less than k, which violates our original restriction that $\varrho_G = k$; or the code would be a rate $1/n$ code that can be trivially put into invariant-factors form). Let $\Delta = GCD(\alpha, \beta)$ be the greatest common divisor of α and β.

The IFD Algorithm finds x and y, both in $F[D]$, such that $\alpha x + \beta y = \Delta$, and does elementary column and/or row interchanges (always unimodular) and then a unimodular transformation (recalling that Δ divides both α and β so $\beta/\Delta \in F[D]$ and $\alpha/\Delta \in F[D]$):

$$\begin{bmatrix} x & y & 0 \\ -\frac{\beta}{\Delta} & \frac{\alpha}{\Delta} & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \alpha & & \\ \beta & & \\ & & \end{bmatrix} = \begin{bmatrix} \Delta & & \\ 0 & & \\ & & \end{bmatrix} \tag{B.339}$$

(the minus sign is needed only for GFq elements where $q > 2$.)

IF Δ divides all other elements in the matrix (for instance if $\Delta = 1$, as it often does), then zero the remainder of the first row and column with

$$\begin{bmatrix} 1 & 0 \\ -\frac{\beta}{\Delta} & 1 \end{bmatrix} \begin{bmatrix} \Delta \\ \beta \end{bmatrix} \quad or \quad [\Delta \;\; \beta] \begin{bmatrix} 1 & -\frac{\beta}{\Delta} \\ 0 & 1 \end{bmatrix}$$

transformations.
OTHERWISE, the above process until this occurs (see also Section B.7.2.1's examples).

Eventually this step will produce a matrix of the form

$$\Gamma_1 = \begin{bmatrix} \gamma_1 & 0 \ldots 0 \\ 0 & \\ \vdots & G_1 \\ 0 & \end{bmatrix} \quad . \tag{B.340}$$

The IFD process repeats this recursively for G_1, then G_2, ... G_k.

The determinant of Γ first k columns is $\prod_{i=1}^{k} \gamma_i$. In order for all A and B matrices to be unimodular, each successive GDC Δ_i must be in any $i \times i$ determinant meaning that this factor must be on the diagonal of Γ. As i increases the GCD can only increase and must necessarily include as a factor the GCD for $i - 1$. Thus, the invariant factors also represent the rations of the successive GCDs. Thus, results 1-6 follow from the construction.**QED**.

It is possible to use Theorem B.7.1 to reduce the computation for smaller matrices, as per the following example:

EXAMPLE B.7.3 [4-state $r = 1/2$ code] Returning to the earlier 4-state $G = [1 + D + D^2 \;\; 1 + D^2]$ example, $\Delta_0 = \Delta_1 = 1$, so $\gamma_1 = 1$:

$$\begin{aligned} G &= \left[1 + D + D^2 \quad 1 + D^2\right] && \text{(B.341)} \\ &= [\,1\,] \cdot [\,1 \quad 0\,] \cdot \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \\ a(D) & b(D) \end{bmatrix} && \text{(B.342)} \\ &= A \cdot \Gamma \cdot B \quad . && \text{(B.343)} \end{aligned}$$

The values of $a(D)$ and $b(D)$ are such that $|B| = 1$. Furthermore, they need not be of degree higher than ν. To constraint$|A| = |B| = 1$, let $a(D) = a_0 + a_1 \cdot D + a_2 \cdot D^2$ and $b(D) = b_0 + b_1 \cdot D + b_2 \cdot D^2$. Then

$$a(D)\left(1 + D^2\right) + b(D)\left(1 + D + D^2\right) = 1 \quad . \tag{B.344}$$

Equating terms in D^0 ... D^4 yields:

$$\begin{aligned}
D^0 &: \quad a_0 + b_0 = 1 \;\text{ or }\; a_0 = \bar{b}_0 \;\; \text{,so let } b_0 = 0 \;\; a_0 = 1 && \text{(B.345)}\\
D^1 &: \quad a_1 + b_0 + b_1 = 0 \;\text{ or }\; a_1 = b_1 \;\; \text{,so let } b_1 = 1 \;\; a_1 = 1 && \text{(B.346)}\\
D^2 &: \quad a_0 + a_2 + b_0 + b_1 + b_2 = 0 \;\text{ or }\; a_2 = b_2 \;\; \text{,so let } b_2 = 0 \;\; a_2 = 0 && \text{(B.347)}\\
D^3 &: \quad a_1 + b_1 + b_2 = 0 \;\; \text{,checks} && \text{(B.348)}\\
D^4 &: \quad a_2 + b_2 = 0 \;\; \text{,checks} && \text{(B.349)}
\end{aligned}$$

So then,

$$a(D) = 1 + D \qquad b(D) = D \tag{B.350}$$

is a valid solution for the B matrix, and

$$G = A \cdot \Gamma \cdot B = [\,1\,] \cdot [\,1 \;\; 0\,] \cdot \left[\begin{array}{cc} 1 + D + D^2 & 1 + D^2 \\ 1 + D & D \end{array}\right] \tag{B.351}$$

This example more rapidly computes the IFD. This can be the case for many situations. However, sometimes the regular method is necessary and clearly would better suit a computer-program implementation.

Rate $1/n$ codes are somewhat trivial for IFD, so another is:

EXAMPLE B.7.4 **[8-state rate 2/3 code revisited]** Continuing with the 8-state, $r = 2/3$, convolutional code of Example B.7.2, $\Delta_0 = \Delta_1 = \Delta_2 = 1$, so $\gamma_1 = \gamma_2 = 1$ for the generator matrix

$$\begin{aligned}
G &= \left[\begin{array}{ccc} 1 & D & 0 \\ D^2 & 1 & D \end{array}\right] && \text{(B.352)}\\
&= \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right] \cdot \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right] \cdot \left[\begin{array}{ccc} 1 & D & 0 \\ D^2 & 1 & D \\ a(D) & b(D) & c(D) \end{array}\right] && \text{(B.353)}
\end{aligned}$$

The quantities $a(D)$, $b(D)$, and $c(D)$ satisfy

$$c(D) + b(D) \cdot D + D\left[c(D) \cdot D^2 + D \cdot a(D)\right] = 1 \quad . \tag{B.354}$$

Let $c(D) = 1$ to satisfy the D^0 constraint. Then

$$b(D) \cdot D + D^3 + D^2 \cdot a(D) = 0 \tag{B.355}$$

Then a solution is $a(D) = 1$ and $b(D) = D^2 + D$, leaving

$$G = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right] \cdot \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right] \cdot \left[\begin{array}{ccc} 1 & D & 0 \\ D^2 & 1 & D \\ 1 & D + D^2 & 1 \end{array}\right] \tag{B.356}$$

The third example encoder cannot be decomposed by IVT directly because that 8-state, $r = 3/4$ encoder had feedback. This section extends the concept of invariant factors before proceeding to decompose G with feedback. Multiplication by $\phi(D)$, as described earlier, clears the denominators (eliminates feedback).

B.7.2.2 Extended Invariant Factors

Let a more general $G(D)$ have rational fractions of polynomials as entries, that is the elements of $G(D)$ are in $F_r(D)$. Then,

$$\varphi \cdot G = A \cdot \tilde{\Gamma} \cdot B \tag{B.357}$$

where φ is the least common multiple of the denominators in G, thus permitting the invariant factors decomposition as shown previously on $\varphi \cdot G$. Then

$$G = A \frac{\tilde{\Gamma}}{\varphi} B = A \cdot \Gamma \cdot B \quad , \tag{B.358}$$

where $\Gamma = \frac{\tilde{\Gamma}}{\varphi}$ is in $F_r(D)$, but A and B are still in $F[D]$. We let

$$\gamma_i = \frac{\alpha_i}{\beta_i} \quad , \tag{B.359}$$

where α_i and β_i are in $F[D]$. Since γ_i divides γ_{i+1}, then α_i divides α_{i+1} and β_{i+1} divides β_i.

EXAMPLE B.7.5 **[Return to Example B.7.1's 8-state 3/4 code with feedback]** Returning to the third earlier example,

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{B.360}$$

Since $u_3(D) = v_4(D)$, the nontrivial portion of the encoder simplifies to have generator

$$G = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{B.361}$$

The LCM of the denominators is $\varphi = 1 + D^3$, so

$$\tilde{G} = \varphi G = \begin{bmatrix} 1+D^3 & 0 & D^2 \\ 0 & 1+D^3 & D \end{bmatrix} \quad . \tag{B.362}$$

The GCD's for $\tilde{G}$ are

$$\tilde{\Delta}_0 = \tilde{\Delta}_1 = 1 \quad , \quad \tilde{\Delta}_2 = 1 + D^3 \quad , \tag{B.363}$$

and

$$\tilde{\gamma}_1 = 1 \quad , \quad \tilde{\gamma}_2 = 1 + D^3 \quad . \tag{B.364}$$

So,

$$\tilde{\Gamma} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix} \tag{B.365}$$

If the bottom row of $\tilde{G}$ is proportional to $1+D^3$, it could be factored. Thus, adding D times the bottom row to the top row and interchanging rows accomplishes this proportionality:

$$\tilde{G}_1 = A_1 \cdot \tilde{G} = \begin{bmatrix} 0 & 1 \\ 1 & D \end{bmatrix} \cdot \tilde{G} = \begin{bmatrix} 0 & 1+D^3 & D \\ 1+D^3 & D(1+D^3) & 0 \end{bmatrix} \quad . \tag{B.366}$$

Then

$$\tilde{G}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ a & b & c \end{bmatrix} \tag{B.367}$$

For unimodular B,

$$D \cdot c + (1+D^3) \cdot c + D \cdot (b + a \cdot D) = 1 \tag{B.368}$$

and $c = 1 + D$, $b = 1$, $a = D(1 + D)$ is a solution. Then, $\tilde{G} = A_1^{-1}\tilde{G}_1$

$$\tilde{G} = \begin{bmatrix} D & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ D(1+D) & 1 & 1+D \end{bmatrix} \tag{B.369}$$

or that

$$G = \begin{bmatrix} D & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ D(1+D) & 1 & 1+D \end{bmatrix} \tag{B.370}$$

The rate 3/4 G with feedback is

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{1+D^3} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1+D^3 & D \\ 0 & 1 & D & 0 \\ 0 & D(1+D) & 1 & 1+D \end{bmatrix} . \tag{B.371}$$

IFD form requires reversal of the first and second diagonal entries to:

$$G = \begin{bmatrix} 0 & 1 & 0 \\ D & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1+D^3 & D \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D & 0 \\ 0 & D(1+D) & 1 & 1+D \end{bmatrix} . \tag{B.372}$$

Scrambler Interpretation of the Invariant Factors Decomposition of G Figure B.37 illustrates the IFD's physical interpretation.

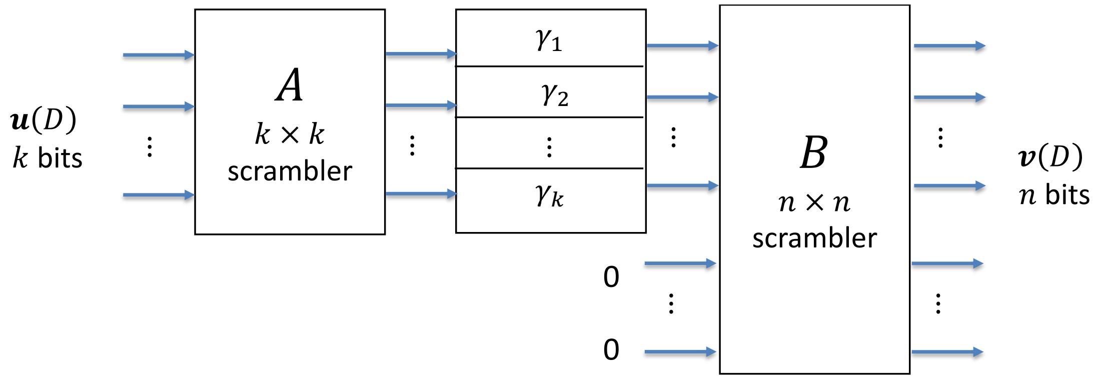


Figure B.37: Scrambler interpretation of IVT.

The input $\boldsymbol{u}_k$ can be any k-dimensional sequence of bits and since A is nonsingular, its output can also be any k-dimensional sequence - all the A matrix does is "scramble" the input bits, This scrambling does otherwise does not affect the code. The matrix Γ matrix also remaps input sequences in 1-to-1 fashion, but does not change the code. Thus a feedback-free encoder sits in the first k rows of B, call it G_b. Additionally, since $B^{-1} \in F[D]$, the first k columns of B^{-1} constitute a feedback-free inverse for G_b.

The effect of additional unitary factors in the matrix B only ffects the input input mapping to the n-dimensional axes of the codewords, but otherwise the most critical code parameters, the distances between codewords and the number of codewords at each distance, remain the same. However, the profile of number of input bit errors b corresponding to distance d $N(b, d)$ can change. As long as the encoder remains noncastasrophic (and $G_b(D)$ will always be so, see upcoming Theorem B.7.2), the change is almost never of practical consequence with noncatasrophic encocders.

More simply put, an MLSD decoder result $\hat{\boldsymbol{u}}(D)$ for noncatastrophic encoder $G(D)$ maps to $\hat{\boldsymbol{v}}(D)$ through $\hat{\boldsymbol{v}}(D) = \hat{\boldsymbol{u}}(D) \cdot G(D)$; which, if the corresponding inputs for another noncastastrophic encoder $G_b(D)$ that corresponds to this same code is desired would be simply $\hat{\boldsymbol{u}}_b(D) = \hat{\boldsymbol{v}}(D) \cdot G_b^{-1}(D)$.

B.7.2.3 Tests for a Noncatastrophic Code:

The following theorem describes three equivalent tests for a noncatastrophic test that derive from the invariant factors decomposition (and avoid searching the state transition diagram for distance-zero loops):

Theorem B.7.2 [Catastrophic Tests] *The following four statements are equivalent:*

1. *An encoder corresponding to a $k \times n$ generator $G(D)$ is noncatastrophic.*
2. *The numerators α_i, $i = 1, ..., k$ of the invariant factors γ_i are of the form D^m, $m \geq 0 \in \mathcal{Z}$, (powers of D).*
3. *The greatest common divisor of the $k \times k$ determinants of $\varphi \cdot G(D)$ is equal to D^δ for some $\delta \geq 0$.*
4. *$G^{-1}(D)$ is feedback free.*

Proof:
(1) ⇒ (2): By contradiction, assume $\alpha_k \neq D^m$, then let $\boldsymbol{u}(D) = \gamma_k^{-1} \cdot e_k \cdot A^{-1}$, where $e_k = [0\ , ...,\ 0\ 1]$, which is an input sequence of necessarily infinite weight. Then, $\boldsymbol{u}(D) \cdot G(D) = e_k \cdot B$ is of finite weight (since $B \in F[D]$), and thus by contradiction $\alpha_k = D^m$.
(2) ⇒ (3): The proof follows directly from the definition of α_i, because these determinants are products of teh α_i.
(3) ⇒ (4): $G^{-1} = B^{-1} \cdot \Gamma^{-1} \cdot A^{-1}$. Since B^{-1}, A^{-1} have all elements in $F[D]$, they are feedback free, and since $\alpha = D^m$, then Γ^{-1} is also feedback free.
(4) ⇒ (1): If we take any finite weight code sequence $\boldsymbol{v}(D)$ and apply to the inverse invariant factors, $\boldsymbol{v}(D) \cdot B^{-1} \cdot \Gamma^{-1} \cdot A^{-1} \in F[D]$, then we must have a corresponding input of finite weight.
QED.

B.7.2.4 Minimal Encoders

A minimal encoder, $G(D)\ \in F[D]$, has a minimum number of delay elements in the obvious realization. A minimal encoder is generated by finding an equivalent encoder that has the property that all its elements are in $F[D]$. A more physical interpretation of the minimal encoder is that it preserves the length (noncatastrophic, finite → finite, infinite → infinite), as well as degree and delay of a sequence. The justification for this assertion will become more clear as this section proceeds.

A precise definition of a **delay-preserving** convolutional encoder follows:

Definition B.7.2 [Delay-Preserving Generator] *A* **delay-preserving generator** *has for any $\boldsymbol{u}(D) \in \{F[D]\}^k$, $del(\boldsymbol{u}) = del(\boldsymbol{u}G)$ for any finite-weight sequence.*

The constant matrix $G_0 = G(0)$ corresponding to the zeroth power of D in $G(D)$ must be nonzero to preserve delay. More formally:

Lemma B.7.1 [Delay Preservation Lemma] *The following are equivalent statements for an encoder $G(D)$ in $F[D]$, or $G(D) = \sum_{m=0}^{\nu} G_m \cdot D^m$:*

1. *For any $\boldsymbol{u}(D) \in \{F[D]\}^k$, $del(\boldsymbol{u}) = del(\boldsymbol{u}G)$. $del(\boldsymbol{u}) = del(\boldsymbol{u}G)\ \forall\ \boldsymbol{u}(D) \in \mathcal{F}^k$.*

2. *The $k \times n$ matrix of constant coefficients, $G_0 = G(0)$, has rank k.*
3. *$G(D)$ has a causal right inverse, $G^{-1}(D)$, in $F[D]$.*

Proof:
(1) ⇒ (2): Proof by contradiction assumes that the rank of G_0 is less than k. Then there exists a constant sequence $\boldsymbol{u}(D) = \boldsymbol{u}_0$ such that $\boldsymbol{u}_0 G_0 = 0$. Then, $0 = del(\boldsymbol{u}_0) \neq del(\boldsymbol{u}_0 G(D)) = 1$, which is a contradiction; so thus G_0 must be of full rank k.
(2) ⇒ (3): The rank-k G_0 augments to an $n \times n$ constant matrix $\bar{G}_0$ that is of rank n by adding appropriate constant rows to the bottom of G_0. Then, these same rows add to the bottom of $G(D)$ to get $\bar{G}(D)$, which is $\bar{G}(D) = \bar{G}_0 + D \cdot \bar{G}'(D)$, where $\bar{G}'(D)$ is some $n \times n$ matrix with entries in $F[D]$. $\bar{G}(D)$ has a simple matrix inverse given by $\bar{G}^{-1}(D) = \left(I_n + D \cdot \bar{G}_0^{-1} \cdot \bar{G}'(D)\right)^{-1} \cdot \bar{G}_0^{-1}$, which is a matrix with no factors of D in the denominator, so it must therefore be causal (starts at time zero or after). Thus, the first k columns of $\bar{G}^{-1}(D)$ form a causal right inverse for $G(D)$. (Note the last $(n-k)$ columns form a causal parity matrix $H(D)$ also).
(3) ⇒ (1): If $G(D)$ has a causal right inverse $G^{-1}(D)$, then $\boldsymbol{u}(D) \cdot G(D) \cdot G^{-1}(D) = \boldsymbol{u}(D)$. Since $G^{-1}(D)$ is causal, it cannot reduce the delay, so that $del(\boldsymbol{u}) = del(\boldsymbol{u}G)$.
QED.

The degree-preserving property of a generator $G(D)$ is similar to delay preservation, but reverses time through

$$\tilde{G}(D^{-1}) \triangleq \begin{bmatrix} D^{-\nu_1} & 0 & ... & 0 \\ 0 & D^{-\nu_2} & ... & 0 \\ . & . & ... & . \\ 0 & 0 & ... & D^{-\nu_k} \end{bmatrix} \cdot G(D) = \sum_{m=0}^{\nu} \tilde{G}_m \cdot D^{-m} \ . \tag{B.373}$$

The quantity ν_i is the same for both $G(D)$ and $\tilde{G}(D^{-1})$, $i = 1, ..., k$, and the entries in $\tilde{G}(D^{-1})$ are in $F[D^{-1}]$, if the entries in $G(D)$ are in $F[D]$.

Definition B.7.3 **[Degree-Preserving Generator]** *A* **degree-preserving generator** *is such that that for* **all** *finite-length $\boldsymbol{u}(D)$,*

$$deg(\boldsymbol{u} \cdot G) = \max_{1 \leq j \leq k} \left[deg(\boldsymbol{u}_j) + \nu_j\right] \quad . \tag{B.374}$$

This leads to a similar reverse-time lemma:

Lemma B.7.2 **[Degree-Preservation Lemma]** *The following are equivalent statements for an encoder $\tilde{G}(D^{-1})$ in $F[D^{-1}]$, for which $\tilde{G}(D) = \sum_{m=0}^{\nu} \tilde{G}_m \cdot D^{-m}$, as above:*

1. *$\tilde{G}(D^{-1})$ preserves the degree of finite-weight sequences, $deg(\boldsymbol{u}) = deg(\boldsymbol{u}\tilde{G})$ $\forall\ \boldsymbol{u}(D) \in \{F[D^{-1}]\}^k$.*
2. *The $k \times n$ matrix of constant coefficients, $\tilde{G}_0$, has rank k.*
3. *$\tilde{G}(D^{-1})$ has an anticausal right inverse, $\tilde{G}^{-1}(D^{-1})$, in $F[D^{-1}]$.*

Proof: Essentially the same as for Lemma B.7.1.

The conditions necessary for a minimal encoder are:

Theorem B.7.3 **[Minimal Encoders]** *An encoder $G(D)$ is minimal, $\nu = \mu$ if and only if all three of the following conditions are met:*

1. *$G(D)$ is delay preserving (or G_0 has rank k).*
2. *$G(D)$ is degree preserving (or $\tilde{G}_0$ has rank k).*
3. *$G(D)$ is non-catastrophic.*

Proof:
First, a minimal encoder must satisfy the theorem's 3 conditions:
condition 1: (by contradiction). If the rank of G_0 is less than k, then $\exists$ $\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 \cdot G_0 = 0$. For those rows of $G(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0$, let j correspond to the the one with largest degree, ν_j. Transformation replaces row j of $G(D)$ by any linear combination of rows in $G(D)$ that includes row j. The linear combination is $D^{-1} \cdot \boldsymbol{u}_0 \cdot G(D)$ reduces the degree of this row and yet still produces an equivalent encoder. Thus, the original encoder could not have been minimal, and by contradiction, $G(D)$ must be delay preserving.

condition 2: (by contradiction). Assume the rank of $\tilde{G}_0$ is less than k, then $\exists$ $\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 \cdot \tilde{G}_0 = 0$. For those rows of $\tilde{G}(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0$, let j correspond to the the one with largest degree, ν_j. Row j in $G(D)$ can be replaced by any linear combination of rows in $G(D)$ that includes a nonzero multiple of row j. An input $\tilde{\boldsymbol{u}}(D) = [u_{k,0} \cdot D^{\nu_j - \nu_1} \; u_{k-1,0} \cdot D^{\nu_j - \nu_2} \; ... \; u_{1,0} \cdot D^{\nu_j - \nu_k}]$ has elements in $F[D]$. The linear combination $\tilde{\boldsymbol{u}}(D) \cdot G(D)$ is an n-vector in $F[D]$ of degree no more than $\nu_j - 1$. The replacement of row j in $G(D)$ by this n-vector reduces the complexity of the encoder $G(D)$, but still maintains an equivalent encoder. Thus, the original encoder could not have been minimal, and by contradiction, $G(D)$ must also be degree preserving.

condition 3: (by contradiction). Let $\boldsymbol{u}(D)$ be some infinitely long sequence that produces a finite-length output $\boldsymbol{v}(D) = \boldsymbol{u}(D) \cdot G(D)$. Then, $\boldsymbol{v}(D) = \sum_{m=r}^{s} \boldsymbol{v}_m \cdot D^m$. If some nonzero elements of $\boldsymbol{u}(D)$ have finite length, then these elements could only affect a finite number of output codewords (since $G(D) \in F[D]$) so that these codewords could not change the output $\boldsymbol{v}(D)$ to be of infinite length (or weight). Thus, this proof can ignore all nonzero elements of $\boldsymbol{u}(D)$ that have finite weight or length. For those rows of $G(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0 = \boldsymbol{u}(D) \mid_{D=0}$, again let j correspond to the the one with largest degree, ν_j. The successive removal of finitely many terms $\boldsymbol{u}_r \cdot D^r$, $\boldsymbol{u}_{r+1} \cdot D^{r+1}, ...\boldsymbol{u}_\tau \cdot D^\tau$ by altering the input sequence produces a corresponding output codeword $\boldsymbol{v}'(D)$ such that $deg(\boldsymbol{v}') < \nu_j$ (that is "the denominator" of $\boldsymbol{u}(D)$ must cancel the "numerator" of $G(D)$ exactly at some point, so that degree is reduced). The replacement of row j by this codeword produces an encoder with lower complexity, without changing the code. Thus, the original encoder could not have been minimal, and by contradiction, a minimal encoder must also be non-catastrophic.

Second, the 3 conditions are sufficient to ensure a minimal encoder. When conditions 1,2, and 3 apply, $len(\boldsymbol{g}_i) = deg(\boldsymbol{g}_i) - del(\boldsymbol{g}_i) + 1 = \nu_i + 1$ ($\boldsymbol{g}_i(D)$ is the row vector corresponding to the i^{th} row of $G(D)$), since $del(\boldsymbol{g}_i) = 0$. Also, $len(\boldsymbol{u}G) = \max_{1 \leq j \leq k} [len(\boldsymbol{u}_j) + \nu_j]$. Since the minimal encoder, call it G_{min}, is equivalent to G, then $G_{min}(D) = A \cdot G(D)$, where A is an invertible matrix with entries in $F[D]$. The ij^{th} entry (row i, column j) is $a_{ij}(D)$, and the i^{th} row of A is $\boldsymbol{a}_i(D)$. Then, for any row i of $G_{min}(D)$

$$len(\boldsymbol{g}_{min,i}) = len(\boldsymbol{a}_i \cdot G) = \max_{1 \leq j \leq k} [len(a_{ij}) + \nu_j] \geq \max_j(\nu_j) \qquad \text{(B.375)}$$

from summing over i, then $\mu \geq \nu$; but since G_{min} is already minimal, $\mu = \nu$, and therefore $G(D)$ is minimal. **QED.**

B.7.2.5 Basic Encoders

Basic encoders are as an intermediate step to deriving a minimal encoder.

Definition B.7.4 [Basic Encoder Properties] *An encoder is a* **basic encoder** *if*

1. *$G(D)$ preserves delay for any $\mathbf{u}(D) \in \mathcal{F}^k$.*
2. *If $\boldsymbol{v}(D) \in (F[D])^n$ and finite weight, then $\mathbf{u}(D) \in (F[D])^k$ and $w_H(\mathbf{u})$ is finite (that is the code is not catastrophic).*

A minimal encoder is necessarily basic. A basic encoder is minimal if the encoder preserves degree. Through the following theorem, a basic encoder has $\alpha_k = 1$, the numerator of the last invariant factor γ_k is one). $\alpha_k = 1$ also implies that $\alpha_i = 1$, $i = 1, ..., k$.

Theorem B.7.4 [Basic Encoders] *An encoder is basic if and only if $\alpha_k = 1$ in the invariant factors decomposition.*
Proof: First, if $\alpha_i = 1$, then the encoder is basic: When $\alpha_i = 1$, then $\gamma_i^{-1} = \beta_i \in F[D]$, and therefore $G^{-1} = B^{-1} \cdot \Gamma^{-1} \cdot A^{-1}$ has entries in $F[D]$ (that is a feedback-free inverse). Then for any finite weight $\boldsymbol{v}(D)$, then $\boldsymbol{v}(D) \cdot G^{-1}(D) = \boldsymbol{u}(D)$ must also be of finite weight, so that the encoder is non-catastrophic. If $G(D)$ is not delay preserving, then there exists a $\boldsymbol{u}_0$ such that $\boldsymbol{u}_0 \cdot G_0 = \boldsymbol{u}_0 \cdot A_0 \cdot \Gamma_0 \cdot B_0 = 0$, and at least one α_i must be zero. Thus, by contradiction, $G(D)$ is delay preserving.

Second, if the encoder is basic, then $\alpha_i = 1$. Proof by contradiction assumes $\alpha_k \neq 1$, and then either $\alpha_k = D^m$ $m > 0$ or $\alpha_k = 1 + \alpha'(D)$ with $\alpha'(D) \neq 0$. In either case, a possible input is $\boldsymbol{u}(D) = \gamma_k^{-1} \epsilon_k A^{-1}$, where $\epsilon_k = [0 \; , ..., 0 \; 1]$. If $\alpha_k = 1 + \alpha'$, then $w_H(\boldsymbol{u}) = \infty$, but $\boldsymbol{u}(D) G(D) = \boldsymbol{b}_k \in F[D]$, so the encoder is catastrophic. Thus, by contradiction in this case, then $\alpha \neq 1 + \alpha'$. If $\alpha_k = D^m$, and noting that the last row of A^{-1} must have at least one term with degree zero (otherwise $|A| \neq 1$), then $del(\boldsymbol{u}) = -m$, $m > 0$. However, the corresponding output $\boldsymbol{b}_k$ can have delay no smaller than 0, so by contradiction (with the delay-preserving condition) $m = 0$. Thus, as both possibilities for $\alpha_k \neq 0$ have been eliminated by contradiction, $\alpha_i = 1$. **QED**.

The following theorem shows that every convolutional encoder is equivalent to a basic encoder.

Theorem B.7.5 [Basic Encoder Equivalent] *Every encoder G is equivalent to a basic encoder.* **Proof:** With IFD $G(D) = A(d) \cdot \Gamma(D) \cdot B(D)$ and the i^{th} row of B as $\boldsymbol{b}_i(D)$, any particular codeword $\boldsymbol{v}(D)$ such that (to avoid confusion, the function of D

notation returns to all quantities in this proof)

$$\begin{aligned} \boldsymbol{v}(D) &= \boldsymbol{u}(D) \cdot G(D) && \text{(B.376)} \\ &= (\boldsymbol{u}(D) \cdot A(D) \cdot \Gamma(D) \cdot B(D) && \text{(B.377)} \\ &= \sum_{i=1}^{n} \left(\boldsymbol{u}(D) \cdot A(D) \cdot \Gamma(D)\right)_i \cdot \boldsymbol{b}_i(D) && \text{(B.378)} \\ &= \sum_{i=1}^{k} \left(\boldsymbol{u}(D) \cdot A(D) \cdot \Gamma(D)\right)_i \cdot \boldsymbol{b}_i(D) && \text{(B.379)} \\ &= \boldsymbol{u}'(D) \cdot G(D) && \text{(B.380)} \end{aligned}$$

since $\left(\boldsymbol{u}(D) \cdot A(D) \cdot \Gamma(D)\right)_i = 0 \; \forall \; i > k$. Since this is true for any $\boldsymbol{v}(D)$, the original code is equivalent to $G'(D)$, a basic encoder with IFD $I \cdot [I \; 0] \cdot B(D)$.

Conversely, G' is equivalent to the original encoder follows from G by writing $\boldsymbol{v}'(D) = \boldsymbol{u}(D) \cdot G' = [\boldsymbol{u}'(D)(\; 0...0] \, B$. Thus $\boldsymbol{v}_1(D) = (\boldsymbol{u}_0(D) \cdot A(D) \cdot \Gamma(D)) \cdot B(D) = \boldsymbol{u}(D) \cdot G(D)$, where $\boldsymbol{u}(D) = \boldsymbol{u}'(D) \cdot \Gamma^{-1}(D) \cdot A^{-1}(D)$. **QED**.

Essentially, the last theorem extracts the top k rows of B in the invariant factors decomposition to obtain a basic equivalent encoder. Also, the last $(n-k)$ columns of B^{-1} form a parity matrix for the code.

B.7.2.6 Construction of Minimal Encoders

Minimal encoder construction follows from the basic encoder is contained within the proof of the following result:

Theorem B.7.6 [Minimal Encoder Construction] *Every basic encoder* $G \in \{F[D]\}^{k \times n}$ *is equivalent to the minimal encoder* G_{min} *through a transformation* A *such that* $|A| = 1$ *(*A *is unimodular).*
Proof: Formation of a minimal encoder from a basic encoder requires the additional property that $\tilde{G}_0$ has rank k. If this property is not yet satisfied by the basic encoder, then there exists f$\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 \cdot \tilde{G}_0 = 0$. Among all rows of $G(D)$ that correspond to nonzero components of $\boldsymbol{u}_0$, let j correspond to the one with largest degree. The replacement of $\boldsymbol{g}_j(D)$ by $\sum_{i=1}^{k} \boldsymbol{u}_{i,0} \cdot D^{\nu_j - \nu_i} \cdot \boldsymbol{g}_i(D)$ produces a new $\boldsymbol{g}_j$ that has degree less than ν_j. This replacement can be performed by a series of row operations with $|A| = 1$. This replacement process continues until the resulting $\tilde{G}_0$ has full rank k. **QED**.

When $\tilde{G}_0$ has full rank k, all $k \times k$ determinants in $\tilde{G}_0$ are nonzero. TheA operations need recording if a final $\hat{\boldsymbol{u}}_{min}(D) \to \hat{\boldsymbol{u}}(D)$ mapping is desired. The Minimal-Encoder Construction Theorem basically states that the maximum degree of all the $k \times k$ determinants of $G(D)$ is equal to μ, when $G(D)$ is minimal. Since these determinants also are invariant to elementary row and column operations, then $\mu =$ max degree of the $k \times k$ determinants in any basic encoder. This also tells us the number of times $(\nu - \mu)$ that the reduction procedure in the previous proof may have to be applied before we obtain a full rank $\tilde{G}_0$. This later result ($k \times k$ determinants) is sometimes quicker to check than forming $\tilde{G}$ and finding $\tilde{G}_0$. The parity matrix forms a dual code, and if we also find the minimal realization of the dual code, then the degree of the minimal generator $H_{min}(D)$, which is also the parity matrix must be the same as the degree of $G_{min}(D)$.

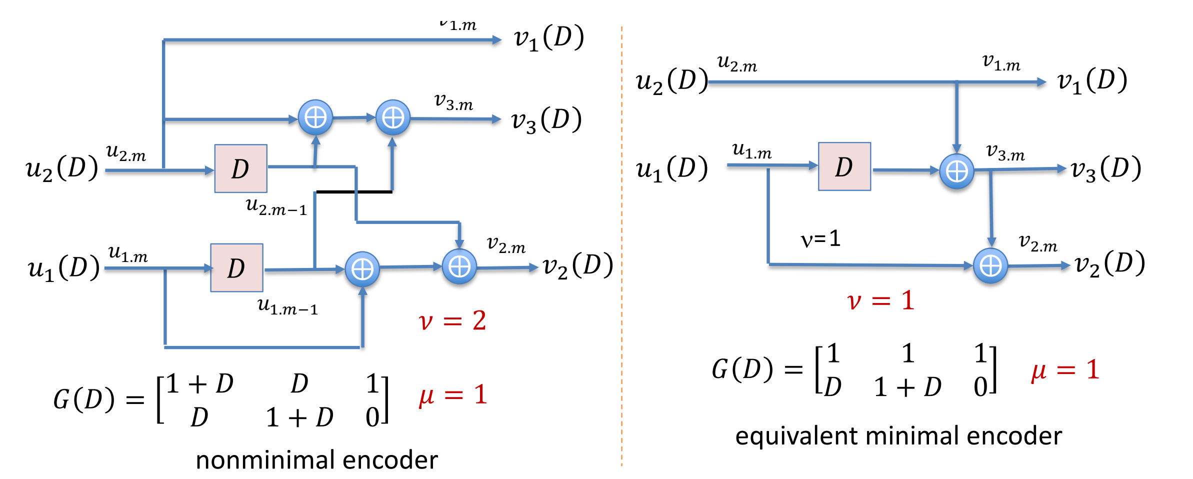


Figure B.38: Example 1 - a non-minimal encoder.

As an example, consider the encoder in Figure B.38. There,

$$G = \begin{bmatrix} 1+D & D & 1 \\ D & 1+D & 0 \end{bmatrix} \quad . \tag{B.381}$$

For this encoder, $\nu = 2$, but that $\mu = 1$ (from rank of $\tilde{G}_0 = 1$, indicating that this basic encoder is not minimal. Thus,

$$\tilde{G} = \begin{bmatrix} 1+D^{-1} & 1 & D^{-1} \\ 1 & 1+D^{-1} & 0 \end{bmatrix} \quad , \tag{B.382}$$

and the linear combination for $\tilde{G}_0$ is $f = [1 \; 1]$ (adding the rows) produces $f \cdot \tilde{G} = [D^{-1} \; D^{-1} \; D^{-1}]$. The replacement of $\tilde{G}$ by

$$\tilde{G} = \begin{bmatrix} D^{-1} & D^{-1} & D^{-1} \\ 1 & 1+D^{-1} & 0 \end{bmatrix} \quad , \tag{B.383}$$

and conversion to G_{min}, by multiplying by D, produces

$$G_{min} = \begin{bmatrix} 1 & 1 & 1 \\ D & 1+D & 0 \end{bmatrix} \quad . \tag{B.384}$$

Figure B.38 shows the corresponding minimum equivalent encoder.

Another example is

$$G = \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \end{bmatrix} \quad , \tag{B.385}$$

which is basic, but $\nu = 4$ and $\mu = 3$. Note this example is illustrated in Appendix **??**, as the alternative "Smith Canonical Form" of Example (**??**) in this chapter. Then,

$$\tilde{G} = \begin{bmatrix} D^{-1} & 1 & 0 \\ 0 & 1+D^{-3} & D^{-2} \end{bmatrix} \quad , \tag{B.386}$$

and again $f = [1 \; 1]$ so that $f\tilde{G} = [D^{-1} \; D^{-3} \; D^{-2}]$ and thus our new $\tilde{G}$ is

$$\tilde{G} = \begin{bmatrix} D^{-1} & 1 & 0 \\ D^{-1} & D^{-3} & D^{-2} \end{bmatrix} \quad , \tag{B.387}$$

leaving a minimal encoder

$$G = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \quad , \tag{B.388}$$

with $\mu = \nu = 3$, which was the original encoder! The two f form

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{B.389}$$

so then

$$\hat{\boldsymbol{u}}(D) = \hat{\boldsymbol{u}}_{min} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad . \tag{B.390}$$

Example B.7.2 had encoder

$$G = \begin{bmatrix} 0 & 1+D^3 & 0 & D \\ 1 & 0 & 0 & 0 \\ 0 & D^4 & 1 & D^2 \end{bmatrix} \quad , \tag{B.391}$$

which is non-minimal with $\nu = 7$ and $\mu = 3$. Here

$$\tilde{G} = \begin{bmatrix} 0 & 1+D^{-3} & 0 & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D^{-4} & D^{-2} \end{bmatrix} \quad , \tag{B.392}$$

and $f = [1\ 0\ 1]$ and $f \cdot \tilde{G} = [0\ D^{-3}\ D^{-4}\ 0]$. The new $\tilde{G}$ is then

$$\tilde{G} = \begin{bmatrix} 0 & 1+D^{-3} & 0 & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & D^{-3} & D^{-4} & 0 \end{bmatrix} \quad , \tag{B.393}$$

(where the factor D^{-3} could be divided from the last row by returning to G through multiplication by D^7, and then realizing that the encoder is still not minimal and returning by multiplying by D^4) and another $f = [1\ 0\ 1]$ with $f\tilde{G} = [0\ D^{-3}\ D^{-1}\ D^{-2}]$. The final $\tilde{G}$ is then

$$\tilde{G} = \begin{bmatrix} 0 & D^{-3} & D^{-1} & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D^{-1} & 0 \end{bmatrix} \quad , \tag{B.394}$$

leaving the final minimal encoder as

$$G = \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \quad , \tag{B.395}$$

for which $\mu = \nu = 3$, which is not the same as our minimal encoder for this example in Section **??**, but is nevertheless equivalent to that encoder.

B.7.2.7 Canonical Systematic Realization

Every encoder is also equivalent to a canonical systematic encoder, where feedback may be necessary to ensure that the encoder realization is systematic. Systematic codes are (trivially) never catastrophic.

The canonical systematic encoder is obtained by following these 3 steps:

1. Find the minimal encoder
2. Every $k \times k$ determinant cannot be divisible by D (otherwise $G_0 = 0$, and the encoder would not be minimal) – find one that is not.
3. Premultiply G_{min} by the inverse of this $k \times k$ matrix.

Example B.7.2 returns so where

$$G = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \quad . \tag{B.396}$$

The first two columns are

$$M = \begin{bmatrix} 1 & D \\ D^2 & 1 \end{bmatrix} \text{ with inverse } M^{-1} = \frac{\begin{bmatrix} 1 & D \\ D^2 & 1 \end{bmatrix}}{1+D^3} \quad , \tag{B.397}$$

so that $G_{sys} = M^{-1} \cdot G_{min}$, or

$$G_{sys} = \frac{1}{1+D^3} \cdot \begin{bmatrix} 1+D^3 & 0 & D^2 \\ 0 & 1+D^3 & D \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \quad . \tag{B.398}$$

The rate 3/4 encoder follows as

$$G = \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \quad . \tag{B.399}$$

With the left-most 3×3 matrix as M, then

$$M^{-1} = \frac{1}{1+D^3} \cdot \begin{bmatrix} 0 & 1+D^3 & 0 \\ 1 & 0 & D^2 \\ D & 0 & 1 \end{bmatrix} \quad . \tag{B.400}$$

Then

$$G_{sys} = \frac{1}{1+D^3} \cdot \begin{bmatrix} 0 & 1+D^3 & 0 \\ 1 & 0 & D^2 \\ D & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \tag{B.401}$$

$$= \frac{1}{1+D^3} \cdot \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D \\ 0 & 0 & 1+D^3 & D^2 \end{bmatrix} \tag{B.402}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D}{1+D^3} \\ 0 & 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} \quad . \tag{B.403}$$

EXAMPLE B.7.6 *[Best 8-state $r = 2/3$ code with feedback, starting only with parity matrix]* Best rate $r = k/(k+1)$ convolutional codes often appear in tables with the simpler specification of their $1/(k+1)$ parity matrices, when $k > 1$. The parity matrix for the best 8-state $r = 2/3$ encoder is

$$H(D) = \left[1+D+D^2+D^3 \quad 1+D^2+D^3 \quad 1+D+D^3\right] \tag{B.404}$$

$$\text{or} \quad \left[\frac{1+D+D^2+D^3}{1+D+D^3} \quad \frac{1+D^2+D^3}{1+D+D^3} \quad 1\right] \quad . \tag{B.405}$$

Then

$$G_{sys} = \begin{bmatrix} 1 & 0 & \frac{1+D+D^2+D^3}{1+D+D^3} \\ 0 & 1 & \frac{1+D^2+D^3}{1+D+D^3} \end{bmatrix} \tag{B.406}$$

Conversion in the straightforward manner of multiplying the systematic $G_{sys}(D)$ matrix by the denominator polynomial creates a basic code, but with excessive complexity.

A^{-1}	G	B^{-1}
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1+D+D^3 & 0 & 1+D+D^2+D^3 \\ 0 & 1+D+D^3 & 1+D^2+D^3 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	row 1 + row 2 → row 1 $\begin{bmatrix} 1+D+D^3 & 1+D+D^3 & D \\ 0 & 1+D+D^3 & 1+D^2+D^3 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	col 1 ↔ col 3 $\begin{bmatrix} D & 1+D+D^3 & 1+D+D^3 \\ 1+D^2+D^3 & 1+D+D^3 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ D^2 & 1+D^2 \end{bmatrix}$	$D^2\cdot$ row 1 + row 2 → row 2 $\begin{bmatrix} D & 1+D+D^3 & 1+D+D^3 \\ 1+D^2 & (1+D^2)(1+D+D^3) & D^2(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ D+D^2 & 1+D+D^2 \end{bmatrix}$	$D\cdot$ row 1 + row 2 → row 2 $\begin{bmatrix} D & 1+D+D^3 & 1+D+D^3 \\ 1+D^2 & (1+D+D^2)(1+D+D^3) & (D+D^2)(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1 & 1 \end{bmatrix}$	row 1 ↔ row 2 $\begin{bmatrix} 1+D^2 & (1+D+D^2)(1+D+D^3) & (D+D^2)(1+D+D^3) \\ D & 1+D+D^3 & 1+D+D^3 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	$D\cdot$ row 1 + row 2 → row 2 $\begin{bmatrix} 1 & (1+D+D^2)\cdot(1+D+D^3) & (D+D^2)\cdot(1+D+D^3) \\ 0 & (1+D+D^2+D^3)\cdot(1+D+D^3) & (1+D^2+D^3)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	$(1+D+D^2)\cdot(1+D+D^3)\cdot$ col 1 + col 2 → col 2 $\begin{bmatrix} 1 & 0 & (D+D^2)\cdot(1+D+D^3) \\ 0 & (1+D+D^2+D^3)\cdot(1+D+D^3) & (1+D^2+D^3)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & (1+D+D^2)\cdot(1+D+D^3) & 0 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	$(1+D+D^2)\cdot(1+D+D^3)\cdot$ col 1 + col 2 → col 2 $\begin{bmatrix} 1 & 0 & (D+D^2)\cdot(1+D+D^3) \\ 0 & (1+D+D^2+D^3)\cdot(1+D+D^3) & (1+D^2+D^3)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & (1+D+D^2)\cdot(1+D+D^3) & (D+D^2)\cdot 1+D+D^3) \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	col 2 + col 3 → col 3 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & (1+D+D^2+D^3)\cdot(1+D+D^3) & (1+D^2+D^3)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & (1+D+D^2)\cdot(1+D+D^3) & 1+D+D^3 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	col 2 + $D^2\cdot$ col 3 → col 2 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & (1+D+D^2)\cdot(1+D+D^3) & (D)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & D^2 & 1 \\ 0 & 1+D^2 & 1 \\ 1 & (1+D)\cdot(1+D+D^3) & 1+D+D^3 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	col 2 + $D\cdot$ col 3 → col 2 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & (1+D)\cdot(1+D+D^3) & (D)\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & D+D^2 & 1 \\ 0 & 1+D+D^2 & 1 \\ 1 & 1+D+D^3 & 1+D+D^3 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	col 2 + col 3 → col 2 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^3 & D\cdot(1+D+D^3) \end{bmatrix}$	$\begin{bmatrix} 0 & 1+D+D^2 & 1 \\ 0 & D+D^2 & 1 \\ 1 & 0 & 1+D+D^3 \end{bmatrix}$
$\begin{bmatrix} D+D^2 & 1+D+D^2 \\ 1+D^2+D^3 & 1+D+D^2+D^3 \end{bmatrix}$	$D\cdot$col 2 + col 3 → col 2 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^3 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1+D+D^2 & 1+D+D^2+D^3 \\ 0 & D+D^2 & 1+D^2+D^3 \\ 1 & 0 & 1+D+D^3 \end{bmatrix}$

The matrix A^{-1} yields through its inversion

$$A = \begin{bmatrix} 1+D+D^2+D^3 & 1+D+D^2 \\ 1+D^2+D^3 & S+D^2 \end{bmatrix} . \tag{B.407}$$

The matrix B^{-1} yields through its inversion

$$B = \begin{bmatrix} (D+D^2)\cdot(1+D+D^3) & (1+D+D^2)\cdot(1+D+D^3) & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{bmatrix} = \begin{bmatrix} D+D^3+D^4+D^5 & 1+D^4+D^5 & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{bmatrix} . \tag{B.408}$$

The basic encoder is in the upper two rows of B, so

$$G_{base}(D) = \begin{bmatrix} D+D^3+D^4+D^5 & 1+D^4+D^5 & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{bmatrix} \tag{B.409}$$

The first format of (B.408) easily allows verification that the basic encoder is orthogonal to the original parity matrix $H(D)$. However, $G_{base}(D)$'s straightforward implementation has

256 states instead of the 8 of $G_{sys}(D)$ (only the last output $v_1(D)$ needs delay elements (3 of them).

Thus the minimization procedure follows:

$A^{-1}_{min,i}$	G_{base}
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D+D^3+D^4+D^5 & 1+D^4+D^5 & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{bmatrix}$
multiply rows by D^{-5} and D^{-3} $\begin{bmatrix} D^{-5} & 0 \\ 0 & D^{-3} \end{bmatrix}$	$\begin{bmatrix} D^{-4}+D^{-2}+D^{-1}+1 & D^{-5}+D^{-1}+1 & D^{-5} \\ D^{-3}+D^{-1}+1 & D^{-3}+D^{-2}+D^{-1}+1 & 0 \end{bmatrix}$
add row1 to row 2 → row 1 $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D^{-4}+D^{-3}+D^{-2} & D^{-5}+D^{-3}+D^{-2} & D^{-5} \\ D^{-3}+D^{-1}+1 & D^{-3}+D^{-2}+D^{-1}+1 & 0 \end{bmatrix}$
multiply rows by D^5 and D^3 $\begin{bmatrix} D^5 & 0 \\ 0 & D^3 \end{bmatrix}$	$\begin{bmatrix} D+D^2+D^3 & 1+D^2+D^3 & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{bmatrix}$
multiply rows by D^{-3} and D^{-3} $\begin{bmatrix} D^{-3} & 0 \\ 0 & D^{-3} \end{bmatrix}$	$\begin{bmatrix} D^{-2}+D^{-1}+1 & D^{-3}+D^{-1}+1 & D^{-3} \\ D^{-3}+D^{-2}+1 & D^{-3}+D^{-2}+D^{-1}+1 & 0 \end{bmatrix}$
add row1 to row 2 → row 2 $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} D^{-2}+D^{-1}+1 & D^{-3}+D^{-1}+1 & D^{-3} \\ D^{-3}+D^{-1} & D^{-2} & D^{-3} \end{bmatrix}$
multiply rows by D^5 and D^3 $\begin{bmatrix} D^3 & 0 \\ 0 & D^3 \end{bmatrix}$	$\begin{bmatrix} D+D^2+D^3 & 1+D^2+D^3 & 1 \\ 1+D & D & 1 \end{bmatrix}$
multiply rows by D^{-3} and D^{-1} $\begin{bmatrix} D^{-3} & 0 \\ 0 & D^{-1} \end{bmatrix}$	$\begin{bmatrix} D^{-2}+D^{-1}+1 & D^{-3}+D^{-1}+1 & D^{-3} \\ D^{-1}+1 & 1 & D^{-1} \end{bmatrix}$
add row1 to row 2 → row 1 $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D^{-2} & D^{-3}+1 & D^{-3}+D^{-1} \\ D^{-1}+1 & 1 & D^{-1} \end{bmatrix}$
multiply rows by D^3 and D $\begin{bmatrix} D^3 & 0 \\ 0 & D \end{bmatrix}$	$\begin{bmatrix} D & 1+D^2 & 1+D^2 \\ 1+D & D & 1 \end{bmatrix}$

The minimal encoder now has 8 states, the minimum complexity.

$$G_{min}(D) = \begin{bmatrix} D & 1+D^2 & 1+D^2 \\ 1+D & D & 1 \end{bmatrix} \quad . \tag{B.410}$$

With some algebra, $G_{min}(D) \cdot H^t(D) = 0$.

There are 3 reduction steps (each with 3 matrix multiplies of $G_{basic}(D)$ above with operations:

$$\begin{aligned}
A_{min,1} &= \begin{bmatrix} D^5 & 0 \\ 0 & D^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D^{-5} & 0 \\ 0 & D^{-3} \end{bmatrix} = \begin{bmatrix} 1 & D^2 \\ 0 & 1 \end{bmatrix} \\
A_{min,2} &= \begin{bmatrix} D^3 & 0 \\ 0 & D^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} D^{-3} & 0 \\ 0 & D^{-3} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
A_{min,3} &= \begin{bmatrix} D^3 & 0 \\ 0 & D \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D^{-3} & 0 \\ 0 & D^{-1} \end{bmatrix} = \begin{bmatrix} 1 & D^2 \\ 0 & 1 \end{bmatrix} \\
A_{min,3} \cdot A_{min,2} \cdot A_{min,1} &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} && \text{(B.411)} \\
A_{min} = I \ . & && \text{(B.412)}
\end{aligned}$$

In this case, the reduction operation did not change input bit-mappings, but this need not be true in general. Further, the the original encoder is

$$\begin{aligned}
G(D) &= A \cdot \Gamma \cdot A_{min} \cdot G_{min}(D) = A \cdot \Gamma \cdot G_{min}(D) && \text{(B.413)} \\
&= \begin{bmatrix} 1+D+D^2+D^3 & 1+D+D^2 \\ 1+D^2+D^3 & D+D^2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^3 & 0 \end{bmatrix} \cdot \begin{bmatrix} D & 1+D^2 & 1+D^2 \\ 1+D & D & 1 \end{bmatrix} \ .
\end{aligned}$$

The MLSD decoder that accepts a channel output and decoders for the input to $G_{min}(D)$ produces $\hat{\boldsymbol{u}}_{min}(D)$. This input sequence can be applied to the encoder $G_{min}(D)$ to generate

$$\hat{\boldsymbol{v}}_{min}(D) = \hat{\boldsymbol{u}}(D) \cdot G_{min}(D) \ . \tag{B.414}$$

Then the inputs desired to the original systematic encoder $G_{sys}(D)$, **which very importantly necessarily was the transmitter's encoder for this to work,** will be then

$$\begin{aligned}
\hat{u}_{sys,2} = \hat{v}_3(D) &= \hat{v}_{min,3}(D) \\
\hat{u}_{sys,1} = \hat{v}_2(D) &= \hat{v}_{min,2}(D) \ .
\end{aligned} \tag{B.415}$$

This exploits that systematic encoders (if, again, actually used) cannot be catastrophic. Since $G_{min}(D)$ is not catastropic, then its MLSD decoder output will have finite input errors if there are finite output channel errors. Further, the generation of $\hat{\boldsymbol{v}}(D)$ is feedback-free using $G_{min}(D)$, so finite input errors lead to finite output errors. The overall system is non-catastrophic. This method allows Viterbi decoder programs (many of which are available publicly but when fed with a systematic feedback encoder for the program to know what structure to decode will increase significantly the number of states they use internally) to be used. While these increased-state programs eventually lead to the same decoded message, it is highly inefficient on programs that already have high run time. Using the same program with $G_{min}(D)$ and the process immediately preceding in (B.414) and (B.415) can lead to several orders of magnitude faster run time to get the same result[7].

[7] At time of writing, the author is not aware of any fixes to the online programs - multiple languages apparently - that all have this unnecessary feedback MLSD complexity increase. The fix does involve finding the minimal encoder one way or another, which as is evident in the table above for a simple $r = 2/3$ encoder was one-design-time significant effort in finite field algebra. A program to fix this awaits a motivated student's interest. Warining: the matlab program smithnorm.m does not appear to accept finite-field inputs, only complex transfer functions. Note that the online programs work with minimum complexity only if the code is rate $r = 1/n$.

Bibliography

[1] John T. Gill, III. *"Stanford EE387 Class Lecture Book"*.

[2] Lasha Ephremidze. *"An Elementary Proof of the Ploynomial Matrix Spectral Factorization Theorem"*. Proceedings of the Royal Society of Edinburgh Section A: Mathematics, DOI: https://doi.org/10.1017/S0308210512001552Published online by Cambridge University Press: 24 July 2014. AMS Subject Classification (2010): 47A68.

Index