ELEC-E7240 Coding Methods D (5 cr)

Patric Östergård

Department of Information and Communications Engineering
Aalto University School of Electrical Engineering

Lectures and Exercises

Lectures: Mondays 12–14 and Wednesdays 10–12

Teacher: Prof. Patric Östergård (Kide, Konemiehentie 1, room 3512), patric.ostergard@aalto.fi

Exercise sessions: Thursdays 14–16

Assistant: Tuomo Valtonen, tuomo.valtonen@aalto.fi

Contents

- ▷ Coding theory
- ▷ Coding and decoding algorithms
- ▷ Application to digital communication

Cryptography is *not* considered in this course.

Prerequisites

Good: ELEC-C7220 Informaatioteoria

More importantly: A good mathematical background (algebra, linear algebra) *or* an interest in mathematics

Literature

The following books are particularly appropriate for students in electrical engineering:

[Wic] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, Upper Saddle River, NJ, 1995. [**Course literature**.]

J. Castiñeira Moreira & P. G. Farrell, *Essentials of Error-Control Coding*, Wiley, Chichester, UK, 2006. [**Course literature** for turbo and LDPC codes.]

S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Second edition, Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, Wiley, Hoboken, NJ, 2005.

Outline of the Course

1. Introduction (1)
2. Algebra, in particular fields and polynomials over fields (2)
3. Linear block codes, cyclic codes (3)
4. BCH and Reed-Solomon codes (1)
5. Convolutional codes, the Viterbi algorithm (2)
6. Modern coding methods, channels with feedback (2)

To pass the course: See separate document

Digital Communication Systems (1)

A digital communication system is a means of transporting information from one party (A) to another (B).

digital The system uses a sequence of symbols from a finite alphabet $(V_q = \{0, 1, \ldots, q-1\})$ to represent the information. Transmission in digital form allows for error control coding.

The basic elements of a digital communication system:

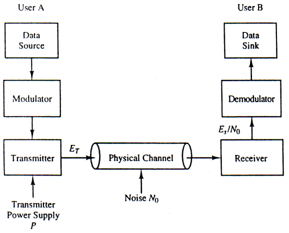


Figure 1-1. Basic elements of a digital communication system

Digital Communication Systems (2)

The modulator maps the information symbols onto signals that can be efficiently transmitted over the communication channel. The selection of a modulation format is a subject that is outside the scope of this course, but is treated in various other ELEC courses.

The physical channel attenuates the transmitted signal and introduces noise. The most commonly assumed noise model is the *additive white Gaussian noise* (AWGN) model.

In environments where noise is present, the demodulated data contains errors. This is usually characterized in terms of a *bit error rate* (BER).

Shannon and Information Theory

In the late 1940s, the work of Shannon and Hamming at Bell Laboratories laid the foundation for error control coding.

Shannon	The father of *information theory* with his paper "A Mathematical Theory of Communication" in 1948. Proved the limits for ideal error control.
Hamming	Presented and analyzed the first practical error control system (based on Hamming codes).

Shannon's results in his seminal paper allows to determine the minimum possible number of symbols necessary for the error-free representation of a given message. A longer message containing the same information is said to have *redundant* symbols.

Codes

On a very general level, **encoding** is about binary relations between two sets A (what is encoded) and B (the result of the encoding). The set B is called a **code**.

The sets A and B are here typically q-ary strings, but any types of sets can be considered in general. For example, B could be a set of images, as in the *Quick Response (QR) code*.

Code Types (1)

source codes Remove *uncontrolled redundancy* and format data.

secrecy codes Encrypt information so that the information cannot be understood by anyone except the intended recipient(s).

error control codes (or channel codes) Format the transmitted information so as to increase its immunity to noise by inserting *controlled redundancy* into the information stream.

Integration of these codes into the basic communication system model is depicted in [Wic, Fig. 1-3].

▷ The order of the three codes is crucial!

Code Types (2)

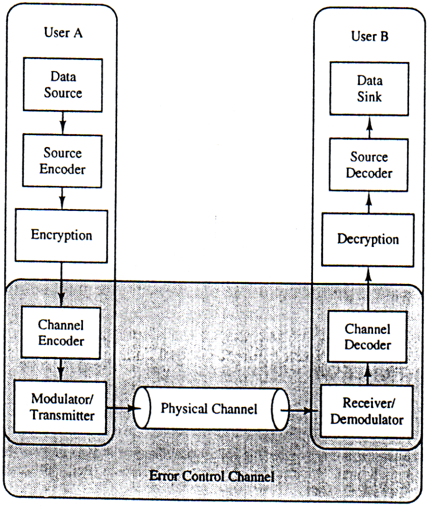


Figure 1-3. Digital communication system coding model

Strategies with Error Control Codes

When error control codes are used, there are several possible ways of reacting to a detected error:

1. Request a retransmission of the erroneous word.
2. Tag the word as being incorrect and pass it along.
3. Attempt to correct the errors in the received word.

The Noisy Channel Coding Theorem

Theorem 1-1. With every channel we can associate a “channel capacity” C. There exist error control codes such that information can be transmitted across the channel at rates less than C with arbitrarily low probability of error (bit error rate of the decoded sequence).

- ▷ The proof of this theorem only shows that such codes exist. It cannot be used to construct good codes in practice!

A Parity-Check Code (1)

Take the output of a binary source and break it up into *k*-bit blocks of the form

$$\mathrm{m} = (m_0, m_1, \ldots, m_{k-1}).$$

At the end of every such block, append a redundant bit *b* as follows to get a codeword (in the rest of this lecture, all additions are carried out modulo 2):

$$\mathrm{c} = (m_0, m_1, \ldots, m_{k-1}, b), \text{ where } b = \sum_{i=0}^{k-1} m_i.$$

A Parity-Check Code (2)

The receiver adds together the values in each coordinate. If the sum is 1, we know that the received word is in error.

$\Rightarrow$ All erroneous words that contain an odd number of errors are detected.

This code is a *single-error-detecting* (or 1-error-detecting) code. A code is said to be *t-error-detecting* if *all* erroneous words with at most t errors are detected.

A Hamming Code (1)

Let the message blocks and codeword be as follows (the length of these is 4 and 7, respectively):

$$\begin{aligned} \mathrm{m} &= (m_0, m_1, m_2, m_3), \\ \mathrm{c} &= (m_0, m_1, m_2, m_3, b_0, b_1, b_2), \\ b_0 &= m_1 + m_2 + m_3, \\ b_1 &= m_0 + m_1 + m_3, \\ b_2 &= m_0 + m_2 + m_3. \end{aligned}$$

A Hamming Code (2)

The received word is denoted by r. The following (binary) values are computed for each such word:

$$\begin{aligned}
\mathrm{r} &= (r_0, r_1, r_2, r_3, r_4, r_5, r_6), \\
s_0 &= r_1 + r_2 + r_3 + r_4, \\
s_1 &= r_0 + r_1 + r_3 + r_5, \\
s_2 &= r_0 + r_2 + r_3 + r_6.
\end{aligned}$$

A Hamming Code (3)

If $s_0 = s_1 = s_2 = 0$, then the received word is a valid word. Otherwise, the value of (s_0, s_1, s_2) gives the position of a single error:

(s_0, s_1, s_2)	Error location
000	None
001	r_6
010	r_5
011	r_0
100	r_4
101	r_2
110	r_1
111	r_3

A Hamming Code (4)

This code is capable of correcting all received words with at most 1 error $\Rightarrow$ it is a *single-error-correcting* code. Analogously, a code is said to be *t-error-correcting* if all erroneous words with at most t errors can be corrected.

Performance Improving with Error Control (1)

Error control is achieved via redundancy. The *code rate* R denotes the ratio of k, the number of data symbols transmitted per codeword, to n, the number of symbols transmitted per codeword. The code in the previous example has rate 4/7.

If we want the data symbol rate R_S (data symbols transmitted per second) to remain constant, the overall symbol rate must be increased to R_S/R.

$\Rightarrow$ If the transmission power level is constant, then the transmitted/received energy per symbol is reduced from E to RE.

Performance Improving with Error Control (2)

The demodulated BER is then increased with respect to its previous value! However, if the code is well selected, the BER at the output of the decoder is better than with the original, uncoded system.

coding gain The additional transmitted power that is required to obtain the same performance without coding.

An example of the coding gain is explained in [Wic, Example 1-2] and depicted in [Wic, Fig. 1-4].

Coding is about saving energy. Green communications!

Performance Improving with Error Control (3)

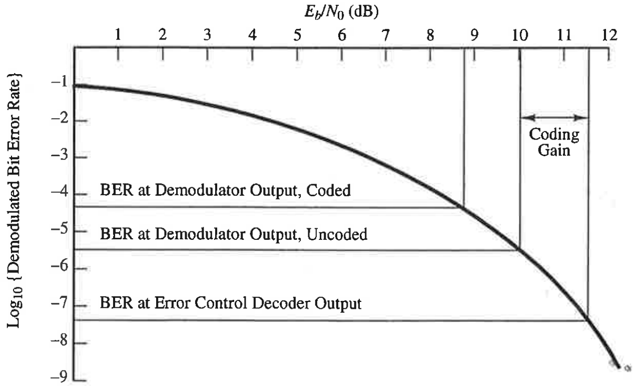


Figure 1-4. Coding gain provided by error control code

Other Applications of Codes

In this course, codes for error-detecting and error-correcting purposes are discussed. In general, a code is any subset of words in a discrete space, and there is a wide variety of possible applications.

Example. A binary *covering* code: $C = \{0000, 0101, 1110, 1011\}$. For any binary word x of length 4, there exists a word in C that differs from x in at most one coordinate. Applications:

- (Lossy) data compression
- Systems for betting (football pools)

Algebra

Mathematics (in particular, algebra) is the language of coding theory. The most important mathematical objects needed in coding theory are *groups*, *finite fields*, and *vector spaces*. The first part of the course is devoted to an in-depth discussion of these topics. (Note: finite field = Galois field.)

Definition: Set

set	An arbitrary collection of elements. A set may be *finite* (e.g., $\{1,2,3\}$), countably infinite (e.g., the positive integers), or uncountably infinite (e.g., the real numbers).
cardinality	The number of objects in the set. The cardinality of a set S is denoted by $|S|$.
order	$=$ cardinality (in particular, when dealing with groups and fields).

So for *sets*, we have cardinality $=$ order $=$ size.

Definition: Group

A **group** is a set G on which a binary operation $\cdot : G \times G \to G$ is defined and for which the following requirements hold:

1. **Associativity:** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$.
2. **Identity:** there exists $e \in G$ such that $a \cdot e = e \cdot a = a$ for all $a \in G$.
3. **Inverse:** for all $a \in G$ there exists an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

A group is said to be *commutative* or *abelian* if it satisfies one more requirement:

4. **Commutativity:** for all $a, b \in G$, $a \cdot b = b \cdot a$.

Examples of Groups

Example 1. The set of integers forms an infinite abelian group under integer addition, but not under integer multiplication (why not?).

Example 2. The set of $n \times n$ matrices with real elements forms an abelian group under matrix addition.

Finite Groups (1)

We are primarily interested in *finite* groups. One of the simplest methods for constructing finite groups lies in the application of modular arithmetic. We write

$$a \equiv b \pmod{m}$$

(pronounced "a is congruent to b modulo m") if $a = b + km$ for some integer k. This relation is reflexive, symmetric, and transitive, and therefore divides the set of integers into m distinct *equivalence classes*.

Finite Groups (2)

Example. Integers modulo 5.
$[0] = \{\ldots, -10, -5, 0, 5, 10, \ldots\}$,
$[1] = \{\ldots, -9, -4, 1, 6, 11, \ldots\}$,
$[2] = \{\ldots, -8, -3, 2, 7, 12, \ldots\}$,
$[3] = \{\ldots, -7, -2, 3, 8, 13, \ldots\}$,
$[4] = \{\ldots, -6, -1, 4, 9, 14, \ldots\}$.

Theorem 2-1. The equivalence classes $[0], [1], \ldots, [m-1]$ form an abelian group of order m under addition modulo m.

Theorem 2-2. The equivalence classes $[1], [2] \ldots, [m-1]$ form an abelian group of order $m-1$ under multiplication modulo m if and only if m is a prime.

The Two Groups of Order 4

·	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Addition mod 4

·	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

A dihedral group

A Multiplicative Group

·	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Multiplication mod 7

More Definitions

order of a group element The order of $g \in G$ is the smallest positive integer n such that $\underbrace{g \cdot g \cdot \cdots \cdot g}_{n} = e$.

subgroup A subset $S \subseteq G$ that forms a group. It is *proper* if $S \neq G$.

Example. The group of addition modulo 4 contains the proper subgroups $\{0\}$ and $\{0, 2\}$.

Definition: Cosets

Let S be a subgroup of G. For any value of $x \in G$, the set $x \cdot S := \{x \cdot s, s \in S\}$ (respectively, $S \cdot x$) forms a **left coset** (respectively, **right coset**) of S in G. If G is abelian, $x \cdot S = S \cdot x$, and left and right cosets coincide and are simply called **cosets**.

Example. The subgroup $\{0, 2\}$ of the group of addition modulo 4 has the cosets $\{0, 2\}$ and $\{1, 3\}$.

Theorem 2-3. The distinct cosets of a subgroup $S \subseteq G$ are disjoint.

Lagrange's Theorem

Theorem 2-4. If S is a subgroup of G, then $|S|$ divides $|G|$.

Proof.

By Theorem 2-3, two distinct cosets of S are disjoint. Moreover, all elements of G belong to some coset of S (for example, an element x belongs to $x \cdot S$). Therefore the distinct cosets, which are of order $|S|$, partition G, and the theorem follows. □

Corollary. A group G of prime order has exactly the following subgroups: $\{e\}$ and G.

Definition: Ring

A **ring** is a set R with two binary operations $\cdot : R \times R \to R$ and $+ : R \times R \to R$ for which the following requirements hold:

1. R forms an abelian group under $+$. The additive identity element is labeled 0.
2. Associativity for $\cdot$: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.
3. The operation $\cdot$ distributes over $+$: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$.

A ring is said to be a **commutative ring** and a **ring with identity**, respectively, if the following two requirements hold:

4. The operation $\cdot$ commutes: $a \cdot b = b \cdot a$.
5. The operation $\cdot$ has an identity element, which is labeled 1.

Examples of Rings

Example 1. The set of integers modulo m under addition and multiplication form a commutative ring with identity.

Example 2. Matrices with integer elements form a ring with identity under standard matrix addition and multiplication.

Example 3. The set of all polynomials with binary coefficients forms a commutative ring with identity under polynomial addition and multiplication with the coefficients taken modulo 2. This ring is denoted GF(2)[x].

Definition: Field

A **field** is a set F with two binary operations $\cdot : F \times F \to F$ and $+ : F \times F \to F$ for which the following requirements hold:

1. F forms an abelian group under $+$. The additive identity element is labeled 0.
2. $F \setminus \{0\}$ forms an abelian group under $\cdot$. The multiplicative identity element is labeled 1.
3. The operations $+$ and $\cdot$ distribute: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

A field can also be defined as a commutative ring with identity in which every non-zero element has a multiplicative inverse.

Examples of Fields

Example 1. The rational numbers form an infinite field.

Example 2. The real numbers form an infinite field, as do the complex numbers.

Example 3. GF(2):

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Constructing Fields

Finite field = Galois field.

Theorem 2-5. Let p be a prime. The integers $\{0, 1, \ldots, p-1\}$ form the field GF(p) under addition and multiplication modulo p.

Theorem. The order of a finite field is p^m, where p is a prime. There is a *unique* field for each such order.

When $m > 1$ in the previous theorem, one cannot use simple modular arithmetic. Instead, such fields can be constructed as vector spaces over GF(p).

Vector Spaces

Let V be a set of *vectors* and F a field of *scalars* with two operations: $+ : V \times V \to V$ and $\cdot : F \times V \to V$. Then V forms a vector space over F if the following conditions are satisfied:

1. V forms an abelian group under $+$.
2. The operations $+$ and $\cdot$ distribute: $a \cdot (\mathrm{u} + \mathrm{v}) = a \cdot \mathrm{u} + a \cdot \mathrm{v}$ and $(a + b) \cdot \mathrm{v} = a \cdot \mathrm{v} + b \cdot \mathrm{v}$.
3. Associativity: For all $a, b \in F$ and all $\mathrm{v} \in V$, $(a \cdot b) \cdot \mathrm{v} = a \cdot (b \cdot \mathrm{v})$.
4. The multiplicative identity $1 \in F$ acts as multiplicative identity in scalar multiplication: for all $\mathrm{v} \in V$, $1 \cdot \mathrm{v} = \mathrm{v}$.

The field F is called the *ground field* of the vector space V.

On Vector Spaces

Example. A vector space over GF(3):
$(1,0,2,1)+(1,1,1,1)=(2,1,0,2)$, $2\cdot(1,0,2,2)=(2,0,1,1)$.

The expression $a_1 \cdot \mathrm{v}_1 + a_2 \cdot \mathrm{v}_2 + \cdots + a_m \cdot \mathrm{v}_m$ where $a_i \in F$, $\mathrm{v}_i \in V$ is called a *linear combination.* A set $\{\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_m\} \subseteq V$ of vectors is called a **spanning set** if all vectors in V can be obtained as a linear combination of these vectors.

A set of vectors is said to be *linearly dependent* if (at least) one of the vectors can be expressed as a linear combination of the others. Otherwise, it is called *linearly independent*.

Basis and Dimension (1)

A spanning set that has minimum cardinality is called a **basis** for V.

Example. The set $\{1000, 0100, 0010, 0001\}$ is a (canonical) basis for V_2^4, where V_q^n denotes the set of q-ary n-tuples.

If a basis for a vector space V has k elements, then it is said to have **dimension** k, written $\dim(V) = k$.

Basis and Dimension (2)

Theorem 2-6. Let $\{\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_k\}$ be a basis for a vector space V. For every vector $\mathrm{v} \in V$ there is a representation $\mathrm{v} = a_1\mathrm{v}_1 + a_2\mathrm{v}_2 + \cdots + a_k\mathrm{v}_k$. This representation is unique.

Corollary. $|V| = |F|^k$.

A vector space V' is said to be a vector **subspace** of V if $V' \subseteq V$.

Inner Product and Dual Spaces

The inner product $\mathrm{u} \bullet \mathrm{v}$ of $\mathrm{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathrm{v} = (v_0, v_1, \ldots, v_{n-1})$ is defined as

$$\mathrm{u} \bullet \mathrm{v} = \sum_{i=0}^{n-1} u_i \cdot v_i.$$

Let C be a k-dimensional subspace of a vector space V. The **dual space** of C, denoted by $C^{\perp}$, is the set of vectors $\mathrm{v} \in V$ such that for all $\mathrm{u} \in C$, $\mathrm{u} \bullet \mathrm{v} = 0$.

Theorem 2-8. The dual space $C^{\perp}$ of a vector subspace $C \subseteq V$ is itself a vector subspace of V.

The Dimension Theorem

Theorem 2-9. Let C be a vector subspace of V. Then $\dim(C) + \dim(C^\perp) = \dim(V)$.

Example. A code $C \subseteq V_2^4$: $C = \{0000, 0101, 0001, 0100\}$, $C^\perp = \{0000, 1010, 1000, 0010\}$. Then $\dim(C) + \dim(C^\perp) = 2+2 = 4 = \dim(V)$.

Question. What is the dual space $C^\perp$ when $C = V$?

Properties of Finite Fields (1)

With $\beta \in \mathrm{GF}(q)$ and 1 the multiplicative identity, consider the sequence

$$1, \beta, \beta^2, \ldots .$$

In a finite field, this sequence must begin to repeat at some point.

The *order* of an element $\beta \in \mathrm{GF}(q)$, written $\mathrm{ord}(\beta)$, is the smallest positive integer m such that $\beta^m = 1$ (cf. order of group element).

Properties of Finite Fields (2)

Theorem 2-10. If $t = \mathrm{ord}(\beta)$, then $t \mid (q-1)$.

Proof.
The set $\{\beta, \beta^2, \ldots, \beta^{\mathrm{ord}(\beta)} = 1\}$ forms a subgroup of the nonzero elements in GF(q) under multiplication. The result then follows from Lagrange's theorem (Theorem 2-4). □

Example. The elements of the field GF(16) can only have orders in $\{1, 3, 5, 15\}$.

The Euler Totient Function

The **Euler** ϕ (or **totient**) **function**, $\phi(t)$, denotes the number of integers in $\{1, 2, \ldots, t-1\}$ that are *relatively prime* to t. This function can be computed as follows when $t > 1$ ($\phi(1) = 1$):

$$\phi(t) = t \prod_{p|t,\ p \text{ prime}} \left(1 - \frac{1}{p}\right).$$

Example 1. $\phi(56) = \phi(2^3 \cdot 7) = 56(1 - 1/2)(1 - 1/7) = 24.$

Example 2. If t is a prime, then $\phi(t) = t(1 - 1/t) = t - 1$, as expected.

Primitive Elements in Finite Fields

▷ If $t \nmid (q-1)$, then there are no elements of order t in GF(q) (Theorem 2-10).

Theorem 2-12. If $t \mid (q-1)$, then there are $\phi(t)$ elements of order t in GF(q).

An element in GF(q) with order $(q-1)$ is called a **primitive element** in GF(q). There are $\phi(q-1)$ primitive elements in GF(q).

⇒ All nonzero elements in GF(q) can be represented as $(q-1)$ consecutive powers of a primitive element.

Example: GF(7)

Order i	Elements of order i	$\phi(i)$
1	$\{1\}$	1
2	$\{6\}$	1
3	$\{2, 4\}$	2
4	None	–
5	None	–
6	$\{3, 5\}$	2

For example, $5^1 = 5$, $5^2 = 4$, $5^3 = 6$, $5^4 = 2$, $5^5 = 3$, $5^6 = 1$.

Characteristic of Field

The **characteristic** of GF(q) is the smallest integer m such that $\underbrace{1+1+\cdots+1}_{m} = 0$.

Theorem 2-13. The characteristic of a finite field is a prime.

Theorem 2-14. The order of a finite field is a power of a prime.

Finite Fields of Order p^m

The following results were discussed in the previous lecture:

- ▷ The order of a finite field is a prime power.
- ▷ There is a unique finite field for each such order.
- ▷ If the order of a finite field is a prime p, one may act on $\{0, 1, \ldots, p - 1\}$ with addition and multiplication modulo p.

The case p^m, $m > 1$, is (somewhat) more complicated. In the sequel, p is always a *prime*.

The Ring of Polynomials $\mathrm{GF}(q)[x]$

The collection of all polynomials $a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ with arbitrary degree and $a_i \in \mathrm{GF}(q)$ is denoted by $\mathrm{GF}(q)[x]$. (Earlier example: these polynomials form a commutative ring with identity.)

Example. We consider $\mathrm{GF}(3)[x]$:

$(x^3 + 2x^2 + 1) + (x^2 + x + 1) = x^3 + 3x^2 + x + 2 = x^3 + x + 2,$

$(x + 1) \cdot (x^2 + 2x + 1) = x^3 + 2x^2 + x + x^2 + 2x + 1 = x^3 + 1.$

Irreducible Polynomials

A polynomial $f(x) \in \mathrm{GF}(q)[x]$ is **irreducible** if $f(x)$ cannot be factored into a product of lower-degree polynomials in $\mathrm{GF}(q)[x]$. Otherwise, it is said to be **reducible**.

Example 1. The polynomial $x^3 + 1 \in \mathrm{GF}(3)[x]$ is not irreducible (a factoring is given in the previous example).

Example 2. The polynomial $x^2 + x + 1 \in \mathrm{GF}(2)[x]$ is irreducible, but $x^2 + x + 1 \in \mathrm{GF}(4)[x]$ is not. (Irreducibility in $\mathrm{GF}(2)[x]$ follows as $x \cdot (x+1) = x^2 + x$, $(x+1) \cdot (x+1) = x^2 + 1$, and $x \cdot x = x^2$.)

$\Rightarrow$ The term *irreducible* must be used only with respect to a specific ring.

Primitive Polynomials

An irreducible polynomial $f(x) \in \mathrm{GF}(p)[x]$ of degree m is **primitive** if the smallest n for which $f(x)$ divides $x^n - 1$ is $n = p^m - 1$. (It can be shown that $f(x)$ always divides $x^{p^m-1} - 1$.)

Example 1. The polynomial $x^3 + x + 1 \in \mathrm{GF}(2)[x]$ is primitive, since it is irreducible and does not divide any of $x^4 - 1$, $x^5 - 1$, and $x^6 - 1$ ($p^m - 1 = 2^3 - 1 = 7$).

NOTE!!! In GF(2), $-1 = 1$.

Example 2. In $\mathrm{GF}(2)[x]$, $x^7 - 1 = x^7 + 1$.

There are $\phi(2^m - 1)/m$ binary primitive polynomials of degree m; for small values of m, see [Wic, Appendix A].

Polynomials Modulo $f(x)$

We denote the ring of polynomials $\mathrm{GF}(q)[x]$ modulo $f(x)$ by $\mathrm{GF}(q)[x]/f(x)$.

Example. In $\mathrm{GF}(3)[x]/(x^2)$, we operate on the polynomials of degree at most 1 (there are nine such polynomials) and with coefficients in GF(3). For example, $x(x+1) = x^2 + x = x$.

Two Methods for Constructing a Field of Order p^m, $m > 1$

Method 1:

If $f(x) \in \mathrm{GF}(p)[x]$ is an irreducible polynomial of degree m, then $\mathrm{GF}(p)[x]/f(x)$ is a field of order p^m.

Method 2:

1. Take a primitive polynomial $f(x)$ of degree m in $\mathrm{GF}(p)[x]$, and let α be a root of $f(x)$ ($f(\alpha) = 0$).
2. The elements of the field are $0, 1 = \alpha^0, \alpha^1, \ldots, \alpha^{p^m-2}$ taken modulo $f(\alpha)$.
3. Carry out addition and multiplication modulo $f(\alpha)$.

We shall now construct GF(4) in these two ways using the polynomial $f(x) = x^2 + x + 1 \in \mathrm{GF}(2)[x]$, which is irreducible as well as primitive.

First Construction of GF(4)

Here the elements of GF(4) are the four polynomials in GF(2)[x] with degree at most 1. Addition comes directly and multiplication is done modulo $f(x) = x^2 + x + 1$.

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\cdot$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Second Construction of GF(4)

Here we let α be a root of the primitive polynomial $f(x) = x^2 + x + 1 \in \mathrm{GF}(2)[x]$, that is $\alpha^2 + \alpha + 1 = 0$. The elements are $0, 1 = \alpha^0, \alpha^1, \alpha^2$, and α is a primitive element of the field. Now multiplication comes directly; when adding we utilize $\alpha^2 + \alpha + 1 = 0$.

$+$	0	$1 = \alpha^0$	α^1	α^2
0	0	α^0	α^1	α^2
$1 = \alpha^0$	α^0	0	α^2	α^1
α^1	α^1	α^2	0	α^0
α^2	α^2	α^1	α^0	0

$\cdot$	0	$1 = \alpha^0$	α^1	α^2
0	0	0	0	0
$1 = \alpha^0$	0	α^0	α^1	α^2
α^1	0	α^1	α^2	α^0
α^2	0	α^2	α^0	α^1

Computing in Fields (1)

A summary about operations in the field GF(p^m):

- ▷ Addition is direct if one considers a ring of polynomials modulo some polynomial.
- ▷ Multiplication is direct if one represent the elements of the field as $0, 1 = \alpha^0, \alpha^1, \ldots, \alpha^{p^m-2}$.
- ▷ **But:** In neither of these cases can the other operation be carried out in a direct way.

Computing in Fields (2)

The study of algorithms for operating on fields is one of most important research topics in computational algebra with many important applications (in coding, cryptography, etc.).

Two possibilities if the field GF(q) is relatively small:

- Construct a look-up table of size $q \times q$.
- Use so-called *Zech logarithms*; this requires a table of size q.

Subfields

A subset $S \subseteq \mathrm{GF}(p^m)$ that is a field is called a **subfield** of $\mathrm{GF}(p^m)$. Every field $\mathrm{GF}(p^m)$ has itself as subfield; any other subfield is called *proper*.

Theorem. The subfields of $\mathrm{GF}(p^m)$ are exactly the fields $\mathrm{GF}(p^a)$ where $a \mid m$.

Example. $\mathrm{GF}(64) = \mathrm{GF}(2^6)$ contains $\mathrm{GF}(2^1)$, $\mathrm{GF}(2^2)$, $\mathrm{GF}(2^3)$, and $\mathrm{GF}(2^6)$ as subfields.

Minimal Polynomials

Let $\alpha \in \mathrm{GF}(q^m)$. The **minimal polynomial** of α with respect to $\mathrm{GF}(q)$ is the smallest-degree nonzero polynomial $f(x) \in \mathrm{GF}(q)[x]$ such that $f(\alpha) = 0$.

Theorem 3-2. For each $\alpha \in \mathrm{GF}(q^m)$ there exists a unique monic polynomial $f(x) \in \mathrm{GF}(q)[x]$ of minimal degree such that

1. $f(\alpha) = 0$,
2. $\deg(f(x)) \leq m$,
3. $g(\alpha) = 0$ implies that $g(x)$ is a multiple of $f(x)$,
4. $f(x)$ is irreducible in $\mathrm{GF}(q)[x]$.

Another definition for *primitive polynomials:* the minimal polynomials for primitive elements in a Galois field.

Conjugates of Field Elements

Motivation: If we want a polynomial $f(x) \in \mathrm{GF}(q)[x]$ to have a root $\alpha \in \mathrm{GF}(q^m)$, what other roots must the polynomial have?

The **conjugates** of $\alpha \in \mathrm{GF}(q^m)$ with respect to the subfield $\mathrm{GF}(q)$ are the elements $\alpha^{q^0} = \alpha, \alpha^{q^1}, \alpha^{q^2}, \ldots$, which form the *conjugacy class of* α *with respect to* $\mathrm{GF}(q)$.

Theorem 3-3. The conjugacy class of $\alpha \in \mathrm{GF}(q^m)$ with respect to $\mathrm{GF}(q)$ contains d elements (that is, $\alpha^{q^d} = \alpha$) with $d \mid m$.

Conjugacy Classes and Roots

Example. By Theorems 2-10 and 2-12, the orders of elements in GF(16) are 1, 3, 5, and 15. Let α be an element of order 3. The conjugates of α with respect to GF(2) are $\alpha, \alpha^2, \alpha^{2^2} = \alpha^3\alpha = \alpha$, so the conjugacy class is $\{\alpha, \alpha^2\}$.

Theorem 3-4. Let $\alpha \in \mathsf{GF}(q^m)$ and let $f(x)$ be the minimal polynomial of α with respect to GF(q). The roots of $f(x)$ are exactly the conjugates of α with respect to GF(q).

Corollary. All the roots of an irreducible polynomial have the same order.

Example: Minimal Polynomials for GF(8)

Let α be root of the primitive polynomial $x^3 + x + 1 \in \mathrm{GF}[2](x)$. Then the elements of GF(8) are

$0 = 0$, $\alpha^0 = 1$, $\alpha^1 = \alpha$, $\alpha^2 = \alpha^2$, $\alpha^3 = \alpha + 1$, $\alpha^4 = \alpha^2 + \alpha$,
$\alpha^5 = \alpha^2 + \alpha + 1$, $\alpha^6 = \alpha^2 + 1$.

Conjugacy class	Minimal polynomial
$\{0\}$	$M_*(x) = x - 0 = x$
$\{\alpha^0 = 1\}$	$M_0(x) = x - 1 = x + 1$
$\{\alpha, \alpha^2, \alpha^4\}$	$M_1(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4) = x^3 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^5\}$	$M_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^5) = x^3 + x^2 + 1$

Factoring $x^n - 1$ (1)

Theorem 3-5. The nonzero elements in GF(q^m) form the complete set of roots of $x^{(q^m-1)} - 1 = 0$.

Proof.

For an arbitrary $\alpha \in$ GF(q^m), ord(α) $\mid (q^m - 1)$ by Theorem 2-10, and therefore α is a root of $x^{(q^m-1)} - 1 = 0$. Moreover, the equation $x^{(q^m-1)} - 1 = 0$ is of degree $(q^m - 1)$ and can therefore have at most $(q^m - 1)$ roots. Therefore, the nonzero elements of GF(q^m) comprise the complete set of roots. $\square$

Example. Factorization of $x^7 - 1$ in GF(2)[x]. Using the results on the previous slide,

$$x^7 - 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Factoring $x^n - 1$ (2)

We now know how to factor $x^{(q^m-1)} - 1$ into irreducible polynomials in the ring $\mathrm{GF}(q)[x]$. What about the general case $(x^n - 1)$?

All roots of $(x^n - 1)$ are nth roots of unity. We need to

1. identify the field where we can find all of these roots,
2. separate the roots into conjugacy classes, and
3. compute the minimal polynomials of the nth roots of unity.

If we have an element $\beta \in \mathrm{GF}(p^m)$ of order n, then the solutions to $x^n - 1 = 0$ are $1, \beta, \beta^2, \ldots, \beta^{n-1}$. These *distinct* elements of order n are often called *primitive nth roots of unity.*

How to find β and the order of the field?

Factoring $x^n - 1$ (3)

By Theorem 2-12, we know that if $n \mid (p^m - 1)$, then there are $\phi(n) > 0$ elements of order n in $\mathrm{GF}(p^m)$.

The **order of q modulo n** is the smallest integer m such that $n \mid (q^m - 1)$.

Example. Factoring $x^5 - 1$ in $\mathrm{GF}(2)[x]$. Since $5 \not| (2^1 - 1)$, $5 \not| (2^2 - 1)$, $5 \not| (2^3 - 1)$, $5 \mid (2^4 - 1)$, GF(16) is the smallest binary extension field in which one may find primitive 5th roots of unity (and the order of 2 modulo 5 is 4).

Example: Factoring $x^{25} - 1$ in GF(2)[x]

Let β be a primitive 25th root of unity. The order of 2 modulo 25 is 20, so we consider GF(2^{20}). The 25 roots of $x^{25} - 1 = 0$ can be grouped into the following conjugacy classes with respect to GF(2):

$$\begin{aligned}
&\{1\},\\
&\{\beta, \beta^2, \beta^4, \beta^8, \beta^{16}, \beta^7, \beta^{14}, \beta^3, \beta^6, \beta^{12}, \beta^{24}, \beta^{23}, \beta^{21}, \beta^{17}, \beta^9, \beta^{18}, \beta^{11},\\
&\beta^{22}, \beta^{19}, \beta^{13}\},\\
&\{\beta^5, \beta^{10}, \beta^{20}, \beta^{15}\}.
\end{aligned}$$

Consequently, $x^{25} - 1$ factors into three irreducible binary polynomials: one of degree one $(x - 1)$, one of degree four, and one of degree twenty.

Cyclotomic Cosets

Cyclotomic cosets provide a simpler framework for what was done in the previous slides.

The **cyclotomic cosets** *modulo n* with respect to GF(q) constitute a partitioning of the integers into sets of the form

$$\{a, aq, aq^2, \ldots, aq^{d-1}\}.$$

Example. Cyclotomic cosets modulo 25 with respect to GF(2) (cf. previous example):

$\{0\}$,
$\{1, 2, 4, 8, 16, 7, 14, 3, 6, 12, 24, 23, 21, 17, 9, 18, 11, 22, 19, 13\}$,
$\{5, 10, 20, 15\}$.

Block Codes

block code	A code C that consists of words of the form $(c_0, c_1, \ldots, c_{n-1})$, where n is the number of *coordinates* (and is said to be the *length* of the code).
q-ary code	A code whose coordinate values are taken from a set (alphabet) of size q (unless otherwise stated, GF(q)).
encoding	Breaking the data stream into blocks, and mapping these blocks onto codewords in C.

The encoding process is depicted in [Wic, Fig. 4-1].

Encoding

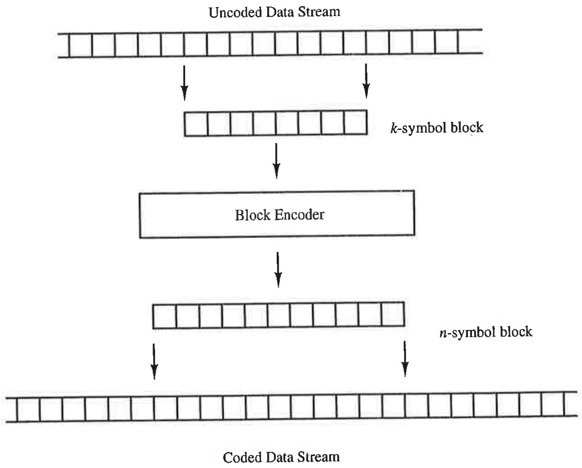


Figure 4-1. Block encoding

Redundancy

If the data blocks of a q-ary code are of length k, then there are $M = q^k$ possible data vectors. (But all data blocks are not necessarily of the same length.)

There are q^n possible words of length n, out of which $q^n - M$ are not valid codewords. The **redundancy** r of a code is

$$r = n - \log_q M,$$

which simplifies to $r = n - k$ if $M = q^k$.

Code Rate

The redundancy is frequently expressed in terms of the code rate. The **code rate** R of a code C of size M and length n is

$$R = \frac{\log_q M}{n}.$$

Again, if $M = q^k$, $R = k/n$.

Transmission Errors (1)

The corruption of a codeword by channel noise, modeled as an additive process, is shown in [Wic, Fig. 4-2].

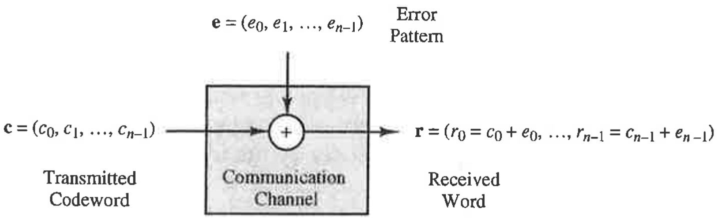

$\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$
Error Pattern
$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$
Transmitted Codeword
+
Communication Channel
$\mathbf{r} = (r_0 = c_0 + e_0, \ldots, r_{n-1} = c_{n-1} + e_{n-1})$
Received Word

Figure 4-2. Baseband model for an additive noisy channel

Transmission Errors (2)

error detection Determination (by the error control decoder) whether errors are present in a received word.

undetectable error An error pattern that causes the received word to be a valid word other than the transmitted word.

error correction Determine which of the valid codewords is most likely to have been sent.

decoder error In error correction, selecting a codeword other than that which was transmitted.

Error Control

The decoder may react to a detected error with one of the following three responses:

automatic repeat request (ARQ) Request a retransmission of the word. For applications where data reliability is of great importance.

muting Tag the word as being incorrect and pass it along. For applications in which delay constraints do not allow for retransmission (for example, voice communication).

forward error correction (FEC) (Attempt to) correct the errors in the received word.

Weight and Distance (1)

The **(Hamming) weight** of a word c, denoted by $w(\mathrm{c})$ (or $w_H(\mathrm{c})$), is the number of nonzero coordinates in c.

Example. $w((0, \alpha^3, 1, \alpha)) = 3$, $w(0001) = 1$.

The *Euclidean distance* between $\mathrm{v} = (v_0, v_1, \dots, v_{n-1})$ and $\mathrm{w} = (w_0, w_1, \dots, w_{n-1})$ is

$$d_E(\mathrm{v}, \mathrm{w}) = \sqrt{(v_0 - w_0)^2 + (v_1 - w_1)^2 + \cdots + (v_{n-1} - w_{n-1})^2}.$$

Weight and Distance (2)

The **Hamming distance** between two words, $\mathrm{v} = (v_0, v_1, \ldots, v_{n-1})$ and $\mathrm{w} = (w_0, w_1, \ldots, w_{n-1})$, is the number of coordinates in which they differ, that is,

$$d_H(\mathrm{v}, \mathrm{w}) = |\{i \mid v_i \neq w_i, 0 \leq i \leq n-1\}|,$$

where the subscript H is often omitted. Note that $w(\mathrm{c}) = d(0, \mathrm{c})$, where 0 is the all-zero vector, and $d(\mathrm{v}, \mathrm{w}) = w(\mathrm{v} - \mathrm{w})$.

The **minimum distance** of a block code C is the minimum Hamming distance between all pairs of distinct codewords in C.

Minimum Distance and Error Detection

Let $d_{\min}$ denote the minimum distance of the code in use. For an error pattern to be undetectable, it must change the values in at least $d_{\min}$ coordinates.

- ▷ A code with minimum distance $d_{\min}$ can detect *all* error patterns of weight less than $d_{\min}$.

Obviously, a large number of error patterns of weight $w \geq d_{\min}$ can also be detected.

Forward Error Correction

The goal in FEC systems is to minimize the probability of decoder error given a received word r. If we know exactly the behavior of the communication system and channel, we can derive the probability $p(\mathrm{c} \mid \mathrm{r})$ that c is transmitted upon receipt of r.

maximum a posteriori decoder (MAP decoder) Identifies the codeword c_i that maximizes $p(\mathrm{c} = \mathrm{c}_i \mid \mathrm{r})$.

maximum likelihood decoder (ML decoder) Identifies the codeword c_i that maximizes $p(\mathrm{r} \mid \mathrm{c} = \mathrm{c}_i)$.

Bayes's rule $p(\mathrm{c} \mid \mathrm{r}) = \frac{p_C(\mathrm{c})p(\mathrm{r}|\mathrm{c})}{p_R(\mathrm{r})}$.

Minimum Distance and Error Correction

The two decoders are identical when $p_C(\mathrm{c})$ is constant, that is, when all codewords occur with the same probability. The maximum likelihood decoder is assumed in the sequel.

The probability $p(\mathrm{r} \mid \mathrm{c})$ equals the probability of the error pattern $\mathrm{e} = \mathrm{r} - \mathrm{c}$. Small-weight error patterns are more likely to occur than high-weight ones $\Rightarrow$ we want to find a codeword that minimizes $w(\mathrm{e}) = w(\mathrm{r} - \mathrm{c})$.

- ▷ A code with minimum distance d_{min} can correct *all* error patterns of weight less than or equal to $\lfloor (d_{\mathrm{min}} - 1)/2 \rfloor$.

It is sometimes possible to to correct errors with $w > \lfloor (d_{\mathrm{min}} - 1)/2 \rfloor$.

Decoder Types

A **complete error-correcting decoder** is a decoder that, given a received word r, selects a codeword c that minimizes $d(\mathrm{r}, \mathrm{c})$.

Given a received word r, a t-**error-correcting bounded-distance decoder** selects the (unique) codeword c that minimizes $d(\mathrm{r}, \mathrm{c})$ iff $d(\mathrm{r}, \mathrm{c}) \leq t$. Otherwise, a *decoder failure* is declared.

Question. What is the difference between decoder errors and decoder failures in a bounded-distance decoder?

Example: A Binary Repetition Code

The binary repetition code of length 4 is $\{0000, 1111\}$.

Received	Selected	Received	Selected
0000	0000	1000	0000
0001	0000	1001	0000 or 1111*
0010	0000	1010	0000 or 1111*
0011	0000 or 1111*	1011	1111
0100	0000	1100	0000 or 1111*
0101	0000 or 1111*	1101	1111
0110	0000 or 1111*	1110	1111
0111	1111	1111	1111

*Bounded-distance decoder declares decoder failure.

Error-Correcting Codes and a Packing Problem

A central problem related to the construction of error-correcting codes can be formulated in several ways:

1. With a given length n and minimum distance d, and a given field GF(q), what is the maximum number $A_q(n, d)$ of codewords in such a code?
2. What is the minimum redundancy for a t-error-correcting q-ary code of length n ?
3. What is the maximum number of spheres of radius t that can be packed in an n-dimensional vector space over GF(q) ?

The Hamming Bound

The number of words in a sphere of radius t in an n-dimensional vector space over GF(q) is

$$V_q(n, t) = \sum_{i=0}^{t} \binom{n}{i} (q-1)^i.$$

Theorem 4-1. The size of a t-error-correcting q-ary code of length n is

$$M \leq \frac{q^n}{V_q(n, t)}.$$

The Gilbert Bound

Theorem 4-2. There exists a t-error-correcting q-ary code of length n of size

$$M \geq \frac{q^n}{V_q(n, 2t)}.$$

Proof.

Repeatedly pick any word c from the space, and after each such operation, delete all words w that satisfy $d(\mathrm{c}, \mathrm{w}) \leq 2t$ from further consideration. Then the final code will have minimum distance at least $2t + 1$ and will be t-error-correcting. The theorem follows from the fact that at most $V_q(n, 2t)$ words are deleted from further consideration in each step. □

Comparing Bounds

Theorems 4-1 and 4-2 say that for the redundancy r of a code,

$$\log_q V_q(n,t) \le r \le \log_q V_q(n,2t).$$

These bounds for binary 1-error-correcting codes are compared in [Wic, Fig. 4-3].

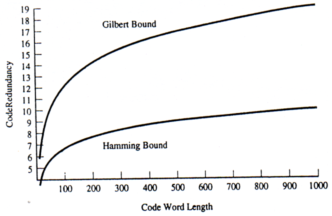


Figure 4-3. A comparison of the Hamming and Gilbert bounds on required redundancy for binary single-error-correcting codes

Perfect Codes

A block code is **perfect** if it satisfies the Hamming bound with equality.

Theorem 4-4. Any nontrivial perfect code over GF(q) must have the same length and cardinality as a Hamming or Golay code.

Note: The sphere packing problem and the error control problem are not entirely equivalent.

List of Perfect Codes

1. $(q, n, k = n, t = 0), (q, n, k = 0, t = n)$: trivial codes.

2. $(q = 2, n \text{ odd}, k = 1, t = (n-1)/2)$: odd-length binary repetition codes (trivial codes).

3. $(q, n = (q^m - 1)/(q-1), k = n - m, t = 1)$ with $m > 0$ and q a prime power: Hamming codes and nonlinear codes with the same parameters.

4. $(q = 2, n = 23, k = 12, t = 3)$: the binary Golay code.

5. $(q = 3, n = 11, k = 6, t = 2)$: the ternary Golay code.

Linear Block Codes

A q-ary code C is said to be **linear** if it forms a vector subspace over GF(q). The *dimension* of a linear code is the dimension of the corresponding vector space.

A q-ary linear code of length n and dimension k (which then has q^k codewords) is called an (n, k) code (or an $[n, k]$ code).

Linear block codes have a number of interesting properties.

Properties of Linear Codes

Property One The linear combination of any set of codewords is a codeword ($\Rightarrow$ the all-zero word is a codeword).

Property Two The minimum distance of a linear code C is equal to the weight of the codeword with minimum weight (because $d(\mathrm{c}, \mathrm{c}') = w(\mathrm{c} - \mathrm{c}') = w(\mathrm{c}'')$ for some $\mathrm{c}'' \in C$).

Property Three The undetectable error patterns for a linear code are independent of the codeword transmitted and always consist of the set of all nonzero codewords.

Generator Matrix

Let $\{\mathrm{g}_0, \mathrm{g}_1, \ldots, \mathrm{g}_{k-1}\}$ be a basis of the codewords of an (n, k) code C over $\mathrm{GF}(q)$. By Theorem 2-6, every codeword $\mathrm{c} \in C$ can be obtained in a unique way as a linear combination of the words g_i. The *generator matrix* G of such a linear code is

$$\mathrm{G} = \begin{bmatrix} \mathrm{g}_0 \\ \mathrm{g}_1 \\ \vdots \\ \mathrm{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix},$$

and a data block $\mathrm{m} = (m_0, m_1, \ldots, m_{k-1})$ is encoded as mG.

Parity Check Matrix

The dual space of a linear code C is called the **dual code** and is denoted by $C^\perp$. Clearly, $\dim(C^\perp) = n - \dim(C) = n - k$, and it has a basis with $n - k$ vectors. These form the **parity check matrix** of C:

$$\mathrm{H} = \begin{bmatrix} \mathrm{h}_0 \\ \mathrm{h}_1 \\ \vdots \\ \mathrm{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{bmatrix}.$$

The Parity Check Theorem

Theorem 4-8. A vector c is in C iff $\mathrm{cH}^T = 0$.

Proof.
($\Rightarrow$) Given a vector $\mathrm{c} \in C$, $\mathrm{c} \bullet \mathrm{h} = 0$ for all $\mathrm{h} \in C^\perp$ by the definition of dual spaces.
($\Leftarrow$) If $\mathrm{cH}^T = 0$, then $\mathrm{c} \in (C^\perp)^\perp$, and the result follows as $(C^\perp)^\perp = C$, which in turn holds as $C \subseteq (C^\perp)^\perp$ and $\dim(C) = \dim((C^\perp)^\perp)$. □

Parity Check Matrix and Minimum Distance

Theorem 4-9. The minimum distance of a code C with parity check matrix H is the minimum nonzero number of columns that has a nontrivial linear combination with zero sum.

Proof.
If the column vectors of H are $\{\mathrm{d}_0, \mathrm{d}_1, \ldots, \mathrm{d}_{n-1}\}$ and
$\mathrm{c} = (c_0, c_1, \ldots, c_{n-1})$, we get
$\mathrm{cH}^T = \mathrm{c}[\mathrm{d}_0\ \mathrm{d}_1\ \cdots\ \mathrm{d}_{n-1}]^T = c_0\mathrm{d}_0 + c_1\mathrm{d}_1 + \cdots + c_{n-1}\mathrm{d}_{n-1}$, so $\mathrm{cH}^T = 0$ is a linear combination of $w(\mathrm{c})$ columns of H. □

Singleton Bound

Theorem 4-10. The minimum distance $d_{\min}$ of an (n, k) code is bounded by $d_{\min} \leq n - k + 1$.

Proof.

By definition, any $r + 1$ columns of a matrix with rank r are linearly dependent. A parity check matrix of an (n, k) code has rank $n - k$, so any $n - k + 1$ columns are linearly dependent, and the theorem follows by using Theorem 4-9. $\square$

Implementing Linear Codes

With linear codes and their generator and parity check matrices, encoding and decoding can be carried out by operating on these matrices (instead of handling complete lists of possible codewords). Very large codes can therefore be handled.

The problem of recovering the data block from a codeword can be greatly simplified through the use of **systematic codes.**

Systematic Codes (1)

Using Gaussian elimination and column reordering it is always possible to get a generator matrix of the form

$$\mathrm{G} = [\mathrm{P} \mid \mathrm{I}_k] = \left[\begin{array}{cccc|cccc} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} & 1 & 0 & \cdots & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & \cdots & 1 \end{array}\right],$$

so that the data block is embedded in the last k coordinates of the codeword: $\mathrm{c} = \mathrm{mG} = [m_0 \ m_1 \ \cdots \ m_{k-1}][\mathrm{P} \mid \mathrm{I}_k] = [c_0 \ c_1 \ \cdots \ c_{n-k-1} \mid m_0 \ m_1 \ \cdots \ m_{k-1}]$.

Systematic Codes (2)

The corresponding parity check matrix for systematic codes is
$\mathrm{H} = [\mathrm{I}_{n-k} \mid -\mathrm{P}^T] =$

$$\left[\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & -p_{0,0} & -p_{1,0} & \cdots & -p_{k-1,0} \\ 0 & 1 & \cdots & 0 & -p_{0,1} & -p_{1,1} & \cdots & -p_{k-1,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -p_{0,n-k-1} & -p_{1,n-k-1} & \cdots & -p_{k-1,n-k-1} \end{array}\right].$$

Standard Array Decoder (1)

A received word r is modeled by the summation $\mathsf{r} = \mathsf{c} + \mathsf{e}$, where c is the transmitted codeword and e is the error pattern induced by the channel noise. The maximum likelihood decoder picks a codeword c' such that $\mathsf{r} = \mathsf{c}' + \mathsf{e}'$, where e' has the smallest possible weight. A look-up table called a **standard array decoder** can be used to implement this process.

Standard Array Decoder (2)

Consider all words in V_q^n in the following way:

1. Remove all codewords in C from V_q^n. List these in a single row, starting with the all-zero word.
2. Select (and remove) one of the remaining words of the smallest weight and write it in the column under the all-zero word. Add this word to all other codewords and write the results in the corresponding columns (and remove these from the set of remaining words).
3. With no remaining words, stop; otherwise, repeat Step 2.

▷ Each row in the table is a coset of C.

Example: Standard Array for a Small Code

With $G = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$, one possible standard array is

0000	1010	1101	0111
0001	1011	1100	0110
0010	1000	1111	0101
0100	1110	1001	0011

Properties of Standard Arrays

- The standard array is uniquely determined exactly when the code is perfect.
- A standard array for a q-ary code of length n has q^n entries, all of which are stored in memory.
- A standard array can be used only for small codes.

The next method to be presented reduces the entry table from size q^n to q^{n-k}.

Syndrome Vectors

For a received vector r, where $r = c + e$, we know that $rH^T = 0$ when $r = c$ ($e = 0$); cf. Theorem 4-8. The matrix product rH^T is called the **syndrome vector** s for the received vector r.

$$\begin{aligned} s &= rH^T \\ &= (c+e)H^T \\ &= cH^T + eH^T \\ &= eH^T \end{aligned}$$

$\Rightarrow$ The syndrome vector depends only on the error pattern. Moreover, the syndrome vector is the same for all words in a row of a standard array (and different for words in different rows).

Example: Syndrome Table for a Small Code

The code used in the previous example has $\mathsf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$.

Error pattern	Syndrome
0000	00
0001	11
0010	10
0100	01

If $\mathsf{r} = 1111$ is received, then $\mathsf{s} = \mathsf{r}\mathsf{H}^T = 10$, so $\mathsf{e} = 0010$ and $\mathsf{c} = 1101$.

Weight Distribution of a Block Code

The weight distribution of an (n, k) code C is a series of coefficients $A_0, A_1, \ldots, A_n$, where A_i is the number of codewords of weight i in C.

The weight distribution is often written as a polynomial $A(x) = A_0 + A_1 x + \cdots + A_n x^n$. This representation is called the **weight enumerator**.

The MacWilliams Identity: Let $A(x)$ and $B(x)$ be the weight enumerators for an (n, k) code C and its $(n, n-k)$ dual code $C^\perp$. Then

$$B(x) = 2^{-k}(1+x)^n A\left(\frac{1-x}{1+x}\right).$$

Binary Hamming Codes

The binary Hamming codes are $(n = 2^m - 1, k = 2^m - m - 1)$ perfect one-error-correcting codes for any integer $m \geq 2$.

The columns of a parity check matrix (of size $m \times n$) of a binary Hamming code consist of all $2^m - 1$ nonzero vectors of length m. The smallest number of such vectors that sum to zero is three $\Rightarrow$ the minimum distance is $d = 3$.

Question. Prove that these codes are indeed perfect.

Decoding Hamming Codes

A received word corrupted by a single error in position i gives $\mathrm{s} = \mathrm{r}\mathrm{H}^T = \mathrm{d}_i^T$, where d_i is the ith column H.

Decoding algorithm for Hamming code:

1. Compute the syndrome $\mathrm{s} = \mathrm{r}\mathrm{H}^T$.
2. Find the column d_i of H that matches the syndrome.
3. Complement the ith bit in the received word.

If the columns of H are in lexicographic order, the decimal value of the syndrome gives the position of the error (with the coordinates numbered $1, 2, \ldots, n = 2^m - 1$).

Weight Enumerator for Hamming Codes (1)

The weight enumerator for the (n, k) binary Hamming code is

$$A(x) = \frac{(1+x)^n + n(1-x)(1-x^2)^{(n-1)/2}}{n+1}.$$

For example, for the $(15, 11)$ binary Hamming code we get $A(x) = 1 + 35x^3 + 105x^4 + 168x^5 + 280x^6 + 435x^7 + 435x^8 + 280x^9 + 168x^{10} + 105x^{11} + 35x^{12} + x^{15}$.

The weight enumerator can be used to calculate exact probabilities of undetected error and decoder error as a function of the binary symmetric channel crossover probability; see [Wic, Fig. 4-9].

Weight Enumerator for Hamming Codes (2)

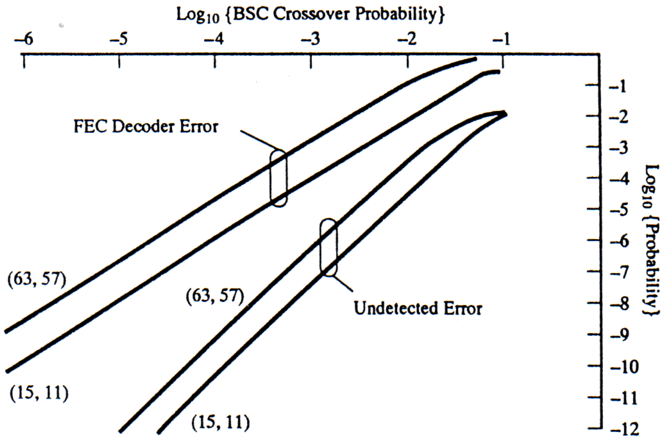


Figure 4-9. Error detection and error correction performance of two Hammin codes

Nonbinary Hamming Codes

Hamming codes over GF(q) are $(n = (q^m - 1)/(q - 1), k = (q^m - 1)/(q - 1) - m)$ perfect one-error-correcting codes for any integer $m \geq 2$.

The column vectors of a parity check matrix (of size $m \times n$) of such a code are selected from the set of $q^m - 1$ nonzero vectors of length m. Since for each such m-tuple, there are $q - 1$ other m-tuples that are multiples of that m-tuple, exactly one m-tuple is selected from each such set of multiples. For example, over GF(3), $(1, 2, 0) + (1, 2, 0) = (2, 1, 0)$.

Modified Codes

puncturing	Delete one of the redundant coordinates. An (n, k) codes becomes an $(n-1, k)$ code.
extending	Add an additional redundant coordinate. An (n, k) code becomes an $(n+1, k)$.
shortening	Delete a message coordinate. An (n, k) code becomes an $(n-1, k-1)$.
lengthening	Add a message coordinate. An (n, k) code becomes an $(n+1, k+1)$ code.

These and two additional terms are illustrated in [Wic, Fig. 4-10].

Linear Cyclic Block Codes (1)

A (linear or nonlinear) code C of length n is said to be **cyclic** if for every codeword $\mathrm{c} = (c_0, c_1, \ldots, c_{n-1}) \in C$, there is also a codeword $\mathrm{c}' = (c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \in C$.

The **code polynomial** of a codeword $c = (c_0, c_1, \ldots, c_{n-1}) \in C$ is $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$. We know that if C is a q-ary (n, k) code, then the codewords form a vector subspace of dimension k within the space of all n-tuples over $\mathrm{GF}(q)$.

Linear Cyclic Block Codes (2)

Let C be a cyclic code, and let $\mathrm{c} = (c_0, c_1, \dots, c_{n-1})$ and c' be two codewords such that c' is obtained by a right cyclic shift of c. Then

$$\begin{aligned} x \cdot c(x) &= x \cdot (c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}) \\ &= c_0 x + c_1 x^2 + \cdots + c_{n-1}x^n \\ &\equiv c_{n-1} + c_0 x + c_1 x^2 + \cdots + c_{n-2}x^{n-1} \pmod{x^n - 1} \\ &\equiv c'(x) \pmod{x^n - 1}. \end{aligned}$$

Now $x^t c(x) \bmod (x^n - 1)$ corresponds to a shift of t places to the right. In general, $a(x)c(x) \bmod (x^n - 1)$, where $a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathrm{GF}(q)[x]/(x^n - 1)$ is an arbitrary polynomial, is a linear combination of cyclic shifts of c and is a codeword.

Properties of Cyclic Codes

Let C be a q-ary (n, k) linear cyclic code.

1. Within the set of code polynomials in C there is a unique monic polynomial $g(x)$ with minimal degree $r < n$ called the *generator polynomial* of C.
2. Every codeword polynomial $c(x) \in C$ can be expressed uniquely as $c(x) = m(x)g(x) \bmod (x^n - 1)$, where $m(x) \in \mathrm{GF}(q)[x]$ is a polynomial of degree less than $n - r$.
3. The generator polynomial $g(x)$ of C is a factor of $x^n - 1$ in $\mathrm{GF}(q)[x]$.

Since $g(x)$ is monic, $g(x) = g_0 + g_1x + \cdots + g_{r-1}x^{r-1} + x^r$.

Question. Why can we assume that $g_0 \neq 0$?

Possible Dimensions of Cyclic Codes (1)

The dimension of a cyclic code C is is $n - r$, where r is the degree of the generator polynomial of C. The factorization of $x^n - 1$ into irreducible polynomials in $\mathrm{GF}(q)[x]$ has been discussed earlier.

Example 1. Binary cyclic codes of length $n = 15$ $(= 2^4 - 1)$. The conjugacy classes formed by the powers of α, an element of order 15 in GF(16) are

$\{1\}$,
$\{\alpha, \alpha^2, \alpha^4, \alpha^8\}$,
$\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9\}$,
$\{\alpha^5, \alpha^{10}\}$,
$\{\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}\}$.

Possible Dimensions of Cyclic Codes (2)

Example 1. (cont.) Hence, the binary polynomial $x^{15} - 1$ factors into one binary polynomial of degree 1, one of degree 2, and three of degree 4. Therefore $x^{15} - 1$ has factors of all degrees between 1 and 15 (for example, $11 = 4 + 4 + 2 + 1$), and there are binary cyclic $(15, k)$ codes for all $1 \le k \le 15$.

Example 2. In a previous lecture, it was shown that the binary polynomial $x^{25} - 1$ factors into one polynomial of degree one, one of degree 4 and one of degree 20. Hence there are binary cyclic $(25, k)$ codes for $k \in \{1, 4, 5, 20, 21, 24, 25\}$.

Encoding Cyclic Codes (1)

Let $g(x)$ be the degree r generator polynomial for an (n,k) q-ary cyclic code C. An $(n-r)$-symbol data block $(m_0, m_1, \ldots, m_{n-r-1})$ is associated with a **message polynomial** $m(x) = m_0 + m_1 x + \cdots + m_{n-r-1}x^{n-r-1}$. Now

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= m_0 g(x) + m_1 x g(x) + \cdots + m_{n-r-1}x^{n-r-1}g(x) \\ &= [m_0 \ m_1 \ \cdots \ m_{n-r-1}] \begin{bmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{n-r-1}g(x) \end{bmatrix}. \end{aligned}$$

Encoding Cyclic Codes (2)

A generator matrix for a cyclic code is then

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & & \\ & g_0 & g_1 & \cdots & g_r & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & & g_0 & g_1 & \cdots & g_r \end{bmatrix},$$

where the unmarked entries are zero.

Decoding Cyclic Codes (1)

Since $g(x) \mid (x^n - 1)$, there exists a **parity polynomial** $h(x)$ such that $g(x)h(x) = x^n - 1$. Moreover, since $g(x) \mid c(x)$, we have that $c(x)h(x) \equiv 0 \pmod{x^n - 1}$. We denote $s(x) := c(x)h(x) \bmod (x^n - 1)$ with $s(x) = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1} \in \mathrm{GF}(q)[x]/(x^n - 1)$. Now

$$\begin{aligned} s(x) &= \sum_{t=0}^{n-1} s_t x^t \equiv c(x)h(x) \equiv \left(\sum_{i=0}^{n-1} c_i x^i\right)\left(\sum_{j=0}^{n-1} h_j x^j\right) \\ &\equiv 0 \pmod{(x^n - 1)} \;\Rightarrow \\ s_t &= \sum_{i=0}^{n-1} c_i h_{(t-i) \bmod n} \end{aligned}$$

Decoding Cyclic Codes (2)

Take the last $(n-k)$ of the parity check equations:

$$\mathrm{s}' = \begin{bmatrix} s_k \\ s_{k+1} \\ \vdots \\ s_{n-1} \end{bmatrix}^T = \begin{bmatrix} \sum_{i=0}^{n-1} c_i h_{(k-i) \bmod n} \\ \sum_{i=0}^{n-1} c_i h_{(k+1-i) \bmod n} \\ \vdots \\ \sum_{i=0}^{n-1} c_i h_{(n-1-i) \bmod n} \end{bmatrix}^T =$$

$$[c_0 \ c_1 \ \cdots \ c_{n-1}] \begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & & & & \\ & h_k & h_{k-1} & \cdots & h_0 & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix}^T = \mathrm{c}\mathrm{H}^T.$$

Decoding Cyclic Codes (3)

By a previous argument, if c is a codeword, then $s' = cH^T = 0$, so the rows of H are vectors in $C^\perp$. Moreover, since the row rank of H is $n - k$ (as $h(x)$ is monic, the rows are linearly independent). Hence the row space spans $C^\perp$, and H is a valid parity check matrix.

Theorem 5-3. Let C be an (n, k) cyclic code with generator polynomial $g(x)$. Then $C^\perp$ is an $(n, n-k)$ cyclic code with generator polynomial $h^*(x)$, the reciprocal of the parity polynomial for C.

Proof.

The parity check matrix has the same structure as the generator matrix. □

Example: Binary Cyclic Code of Length 7 (1)

First, we need to factor $x^7 - 1$ over GF(2)[x]. Let α be a root of $p(x) = 0$, where $p(x)$ is the primitive polynomial $x^3 + x + 1$. The conjugacy classes and the corresponding polynomials are as follows:

$$\begin{array}{lcl} \{1\} & \leftrightarrow & x+1, \\ \{\alpha, \alpha^2, \alpha^4\} & \leftrightarrow & x^3+x+1, \\ \{\alpha^3, \alpha^6, \alpha^5\} & \leftrightarrow & x^3+x^2+1. \end{array}$$

The polynomial $g(x) = (x^3 + x + 1)(x + 1) = x^4 + x^3 + x^2 + 1$ is one possible generator polynomial. The corresponding parity polynomial is $h(x) = (x^7 + 1)/g(x) = x^3 + x^2 + 1$.

Example: Binary Cyclic Code of Length 7 (2)

(cont.) The message polynomials consist of all binary polynomials of degree less than or equal to 2. The code is a $(7, 3)$ code (with $2^3 = 8$ words). A codeword of the code is, for example, $(x^2 + 1) \cdot g(x) = 1 + x^3 + x^5 + x^6 \rightarrow 1001011$. The following matrices are, respectively, a generator matrix and a parity check matrix of the code:

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}, \ H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Systematic Cyclic Codes

Polynomial multiplication encoding for cyclic linear codes is easy. Unfortunately, the codes obtained are in most cases not *systematic*. Systematic cyclic codes can be obtained through a procedure that is only slightly more complicated than the polynomial multiplication procedure.

Systematic Encoding

Consider an (n, k) cyclic code C with generator polynomial $g(x)$. The k-symbol message block is given by the message polynomial $m(x)$.

Step 1. Multiply the message polynomial $m(x)$ by x^{n-k}.

Step 2. Divide the result of Step 1 by the generator polynomial $g(x)$. Let $d(x)$ be the remainder.

Step 3. Set $c(x) = x^{n-k}m(x) - d(x)$.

This encoding works, as (1) $c(x)$ is a multiple of $g(x)$ and therefore a codeword, (2) the first $n - k$ coefficients of $x^{n-k}m(x)$ are zero, and (3) only the first $n - k$ coefficients of $-d(x)$ are nonzero (the degree of $g(x)$ is $n - k$).

Example: Systematic Encoding (1)

We consider the $(7, 3)$ binary cyclic code with generator polynomial $g(x) = x^4 + x^3 + x^2 + 1$ discussed in a previous example, and encode $101 = 1 + x^2 = m(x)$.

Step 1. $x^{n-k}m(x) = x^4(x^2 + 1) = x^6 + x^4$.

Step 2. $x^6 + x^4 = (x^4 + x^3 + x^2 + 1)(x^2 + x + 1) + (x + 1)$, so $d(x) = x + 1$ (*Carry out the necessary division in the same way as you learnt in elementary school!*).

Step 3. $c(x) = x^6 + x^4 - (x + 1) = 1 + x + x^4 + x^6$, and the transmitted codeword is 1100**101**.

Example: Systematic Encoding (2)

The systematic generator matrix is obtained by selecting as rows the codewords associated with the messages 100, 010, and 001. The parity check matrix is obtained using the basic result (presented earlier) that $\mathrm{H} = [\mathrm{I}_{n-k} \mid -\mathrm{P}^T]$ with $\mathrm{G} = [\mathrm{P} \mid \mathrm{I}_k]$. In the current example, we get

$$\mathrm{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right], \ \mathrm{H} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right].$$

Implementations of Cyclic Codes

Data rates are very high in many applications $\Rightarrow$
only very fast decoders and encoders can be used.

Fast circuits are, for example, simple exclusive OR (XOR) gates, switches, and shift registers. For nonbinary encoders and decoders, finite-field adder and multiplier circuits are needed. We now focus on *shift-register* (SR) encoders and decoders for cyclic codes.

Operational Elements in Shift Registers (1)

The symbology used is depicted in [Wic, Fig. 5-1].

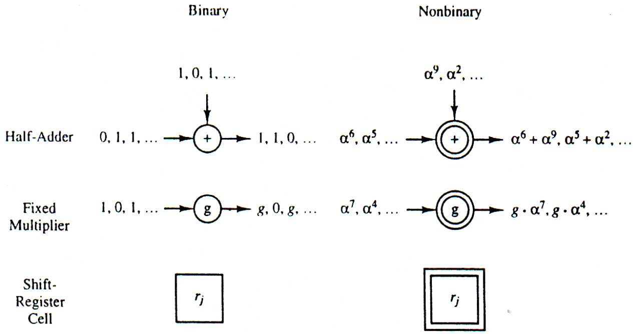

Binary
Nonbinary
Half-Adder
1, 0, 1, ...
0, 1, 1, ...
+
1, 1, 0, ...
$\alpha^9, \alpha^2, \ldots$
$\alpha^6, \alpha^5, \ldots$
$\alpha^6 + \alpha^9, \alpha^5 + \alpha^2, \ldots$
Fixed Multiplier
1, 0, 1, ...
g
g, 0, g, ...
$\alpha^7, \alpha^4, \ldots$
$g \cdot \alpha^7, g \cdot \alpha^4, \ldots$
Shift-Register Cell
r_j
r_j

Figure 5-1. Shift-Register Operational Elements

Operational Elements in Shift Registers (2)

half-adder	Adds the input values without carry. In the binary case, XOR.
SR cell	Flip-flops. In the binary case, one.
fixed multiplier	Multiplies the input value with a given value. In the binary case, existence or absence of connection.

In the nonbinary case, we assume that the field is a *binary* extension field: $\mathrm{GF}(p^m)$ with $p = 2$. The circuits are substantially more complicated when $p \neq 2$.

Addition and SRs in Extension Fields

Elements $\alpha, \beta \in \mathrm{GF}(2^m)$ are represented as binary m-tuples $(a_0, a_1, \ldots, a_{m-1})$ and $(b_0, b_1, \ldots, b_{m-1})$, respectively.

Then the addition of α and β gives $(a_0 + b_0, a_1 + b_1, \ldots, a_{m-1} + b_{m-1})$, where $+$ is binary addition. The nonbinary addition circuit is shown in [Wic, Fig. 5-2].

The non-binary shift-register cells are implemented with one flip-flop for each coordinate in the m-tuple; see [Wic, Fig. 5.4].

Multiplication in Extension Fields

As an example, we consider multiplication in GF(2^4) of an arbitrary value $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ by a *fixed* value $g = 1 + \alpha$, where α is a root of the primitive polynomial $x^4 + x + 1$. Then

$$\begin{aligned}
\beta \cdot g &= (b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3)(1+\alpha) \\
&= b_0 + (b_0 + b_1)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3 + b_3\alpha^4 \\
&= b_0 + (b_0 + b_1)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3 + b_3(\alpha + 1) \\
&= (b_0 + b_3) + (b_0 + b_1 + b_3)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3.
\end{aligned}$$

The corresponding multiplier circuit is illustrated in [Wic, Fig. 5-3].

Nonsystematic Encoders

With message polynomial $m(x) = m_0 + m_1x + \cdots + m_{k-1}x^{k-1}$ and generator polynomial $g(x)$, the codeword polynomial is

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= m_0g(x) + m_1xg(x) + \cdots + m_{k-1}x^{k-1}g(x). \end{aligned}$$

The corresponding SR circuit is shown in [Wic, Fig. 5-5].

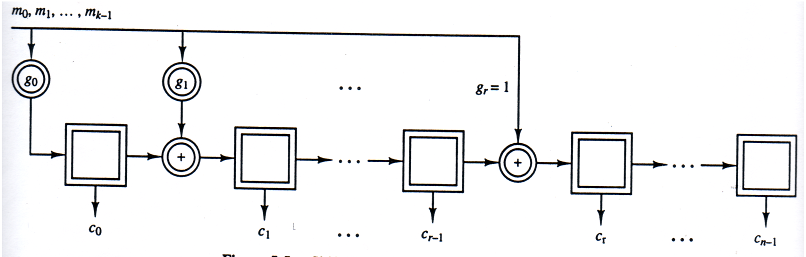


Figure 5-5. Shift-Register Encoder (Nonsystematic)

Systematic Encoders

Step 1. (Multiply $m(x)$ by x^{n-k}.) Easy, shown in [Wic, Fig. 5-8].

Step 2. (Divide the result of Step 1 by $g(x)$, and let $d(x)$ be the remainder.) Polynomial division is carried out through the use of a linear feedback shift register (LFSR) as shown in [Wic, Fig. 5-9], where $a(x)$ is divided by $g(x)$, and $q(x)$ and $d(x)$ are the quotient and remainder, respectively.

Step 3. (Set $c(x) = x^{n-k}m(x) - d(x)$.) Achieved by combining the two SR circuits for the previous steps, as shown in [Wic, Fig. 5-12].

An alternative encoder for cyclic codes, not considered here, is presented in [Wic, Fig. 5-13].

Shift-Register Polynomial Division

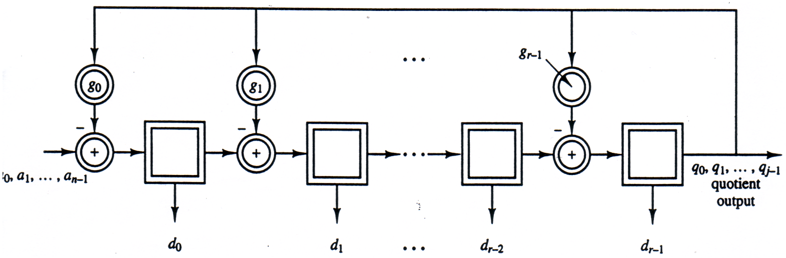


Figure 5-9. Shift-Register Division of $a(x)$ by $g(x)$

An Example of Polynomial Division

Example 5-7—Shift-Register Polynomial Division

Consider the division of the binary polynomial $x^6 + x^4$ by $x^4 + x^3 + x^2 + 1$. The circuit in Figure 5-10 is obtained using the general form in Figure 5-9. The contents of the SR cells at every step of the operation are shown in Figure 5-11.

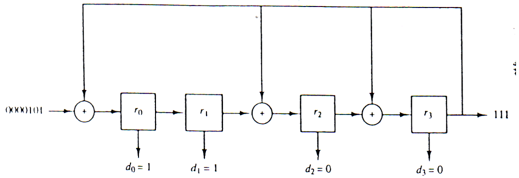
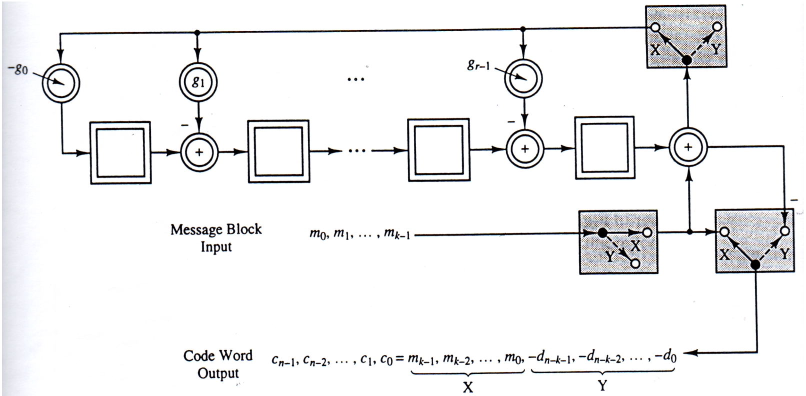


Figure 5-10. Shift-Register Division of $x^6 + x^4$ by $x^4 + x^3 + x^2 + 1$

SR cells	r_0	r_1	r_2	r_3	
Initial state	0	0	0	0	
Input $a_6 = 1$	1	0	0	0	
Input $a_5 = 0$	0	1	0	0	
Input $a_4 = 1$	1	0	1	0	
Input $a_3 = 0$	0	1	0	1	
Input $a_2 = 0$	1	0	0	1	
Input $a_1 = 0$	1	1	1	1	
Input $a_0 = 0$	1	1	0	0	
Final state $= r$	1	1	0	0	$\Leftrightarrow d(x) = x + 1$

Figure 5-11. Shift-Register Cell Contents During Division of $x^6 + x^4$ by $x^4 + x^3 + x^2 + 1$

A Systematic Encoder



$-g_0$
g_1
g_{r-1}
$-$
$-$
$-$
X
Y
Message Block Input
$m_0, m_1, \ldots, m_{k-1}$
Code Word Output
$c_{n-1}, c_{n-2}, \ldots, c_1, c_0 = \underbrace{m_{k-1}, m_{k-2}, \ldots, m_0}_{X}, \underbrace{-d_{n-k-1}, -d_{n-k-2}, \ldots, -d_0}_{Y}$

Figure 5-12. Systematic Encoder for Cyclic Codes

Error Detection for Systematic Codes

The transmitted codeword of a systematic cyclic code has the form

$$\mathrm{c} = (c_0, c_1, \ldots, c_n) = (\underbrace{-d_0, -d_1, \ldots, -d_{n-k-1}}_{\text{remainder block}}, \underbrace{m_0, m_1, \ldots, m_{k-1}}_{\text{message block}}).$$

Error detection is performed on a received word r as follows.

1. Denote the values in the message and parity positions of the received word r by m and d, respectively.
2. Encode m using an encoder identical to that used by the transmitter, and denote the remainder block obtained in this way by d′.
3. Compare d with d′. If they are different, then the received word contains errors.

Syndrome Computation for Systematic Codes

Denote the received word by r with m and d in the message and parity positions, respectively. Let d' be a valid parity block of message m (cf. previous slide), and denote this valid word by r'.

$$\begin{aligned} \mathrm{s} &= \mathrm{r}\mathrm{H}^T \\ &= (\mathrm{r}-\mathrm{r}')\mathrm{H}^T \text{ (as } \mathrm{r}'\mathrm{H}^T = 0) \\ &= (\underbrace{d_0 - d_0', d_1 - d_1', \ldots, d_{n-k-1} - d_{n-k-1}'}_{\mathrm{d}-\mathrm{d}'}, 0, 0, \ldots, 0)\mathrm{H}^T \\ &= \mathrm{d} - \mathrm{d}', \end{aligned}$$

since the parity check matrix has the form $\mathrm{H} = [\mathrm{I}_{n-k} \mid -\mathrm{P}^T]$.

Syndromes for nonsystematic codes can also be computed through the use of shift registers.

Error-Correction Approaches

Error correction has earlier been discussed for general linear codes.

- ▷ A *standard array* has q^n entries.
- ▷ A *syndrome table* has q^{n-k} entries.
- ▷ We shall see that the number of entries of a syndrome table for cyclic linear codes can be reduced to approximatively q^{n-k}/n.
- ▷ With more (algebraic) structure of the codes, even more powerful decoding is possible (to be discussed in forthcoming lectures).

Syndrome Decoding for Cyclic Codes

Theorem 5-3. Let $s(x)$ be the syndrome polynomial corresponding to a received polynomial $r(x)$. Let $r_i(x)$ be the polynomial obtained by cyclically shifting the coefficients of $r(x)$ i steps to the right. Then the remainder obtained when dividing $xs(x)$ by $g(x)$ is the syndrome $s_1(x)$ corresponding to $r_1(x)$.

Having computed the syndrome s with an SR division circuit, we get $s_i(x)$ after the input of i 0s into the circuit! We then need only store one syndrome s for an error pattern e and all cyclic shifts of e.

Decoding Algorithm for Cyclic Codes

1. Let $i := 0$. Compute the syndrome s for a received vector r.
2. If s is in the syndrome look-up table, goto Step 6.
3. Let $i := i + 1$. Enter a 0 into the SR input, computing s_i.
4. If s_i is not in the syndrome look-up table, goto Step 3.
5. Let e_i be the error pattern corresponding to the syndrome s_i. Determine e by cyclically shifting e_i i times to the left.
6. Let $\mathrm{c} := \mathrm{r} - \mathrm{e}$. Output c.

Example: Error Correction of (7,4) Cyclic Code (1)

Consider the (7,4) binary cyclic code generated by $g(x) = x^3 + x + 1$, with parity check polynomial $h(x) = (x^7 + 1)/g(x) = x^4 + x^2 + x + 1$, and with parity check matrix

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

This is a one-error-correcting Hamming code, so all correctable error patterns are cyclic shifts of 0000001. An SR error-correction circuit for this code is displayed in [Wic, Fig. 5-14].

Example: Error Correction of (7,4) Cyclic Code (2)

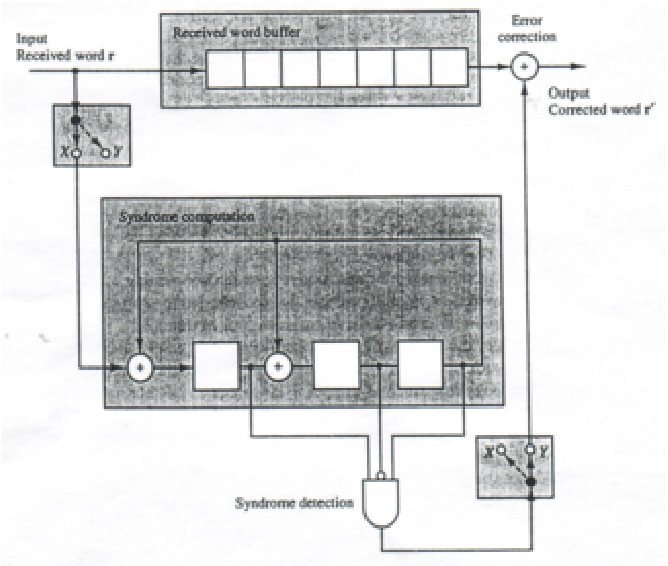


Figure 5-14. Shift-Register Decoder for (7, 4) Cyclic Code

Error Detection in Practice

The most frequently used error control techniques in the history of computers and communication networks are:

one-bit parity check	Very simple, but yet important.
CRC codes	Shortened cyclic codes that have extremely simple and fast encoder and decoder implementations.

Properties of CRC Codes

- **Cyclic redundancy check (CRC)** codes are shortened cyclic codes obtained by deleting the j rightmost coordinates in the codewords.
- CRC codes are generally not cyclic.
- CRC codes can have the same SR encoders and decoders as the original cyclic code.
- CRC codes have error detection and correction capabilities that are at least as good as those of the original cyclic code.
- CRC codes have good burst-error detection capabilities.

Some Generator Polynomials

CRC-4	$g_4(x) = x^4 + x^3 + x^2 + x + 1$
CRC-12	$g_{12}(x) = (x^{11} + x^2 + 1)(x + 1)$
CRC-ANSI	$g_A = (x^{15} + x + 1)(x + 1)$
CRC-CCITT	$g_C = (x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1) \cdot (x + 1)$

Example. The polynomial $g_{12}(x)$ divides $x^{2047} - 1$ but no polynomial $x^m - 1$ with smaller degree, so it defines a cyclic code of length 2047 and dimension $2047 - 12 = 2035$. So, CRC-12 encodes up to 2035 message bits, generating 12 bits of redundancy.

Error Detection Performance Analysis

The error detection performance of codes depends on the type of errors. In performance analysis, the following three situations are most often considered.

1. *Total corruption of words.*
2. *Burst errors.* These are errors that occur over several *consecutive* transmitted symbols.
3. *The binary symmetric channel.*

Total Corruption of Words

When an (n, k) code is used, total corruption leads to a decoder error with probability

$$\frac{q^k}{q^n} = q^{k-n}.$$

Note that this probability is solely a function of the number of redundant symbols in the transmitted codewords.

Example. With CRC-12, an error is detected with probability $1 - 2^{-12} \approx 0.999756$ in case of total corruption.

Burst-Error Detection

A burst-error pattern of length b starts and ends with nonzero symbols; the intervening symbols may be take on any value, including zero.

Theorems 5-4, 5-5, and 5-6. A q-ary cyclic or shortened cyclic codes with generator polynomial $g(x)$ of degree r can detect *all* burst error patterns of length r or less; the fraction $1 - q^{1-r}/(q-1)$ of burst error patterns of length $r+1$; and the fraction $1 - q^{-r}$ of burst error patterns of length greater than $r+1$.

Example. With CRC-12, all bursts of length at most 12, 99.95% of bursts of length 13, and 99.976% of longer bursts are detected.

The Binary Symmetric Channel

An exact determination of the performance of a CRC code over the binary symmetric channel requires knowledge of the weight distribution of the code.

Background

The algebraic structure of linear codes and, in particular, cyclic linear codes, enables efficient encoding and decoding algorithms and fast implementations.

BCH (from the names of Bose, Ray-Chaudhuri, and Hocquenghem) and **Reed-Solomon** codes are even more powerful algebraic codes. Reed-Solomon codes can be described as certain nonbinary BCH codes (they are, however, discussed separately, as Reed-Solomon codes have some interesting properties that are not found in other BCH codes).

Minimum Distance of Cyclic Codes

When constructing an arbitrary cyclic code, there is no guarantee as to the resulting minimum distance. An exhaustive computer search is often needed to find the minimum-weight codewords of a linear code and thereby the minimum distance.

BCH codes, on the other hand, take advantage of a useful result that ensures a lower bound on the minimum distance given a particular constraint on the generator polynomial. This result is known as the *BCH bound*.

The BCH Bound

Theorem 8-1. Let C be a q-ary (n, k) cyclic code with generator polynomial $g(x)$. Let m be the order of q modulo n ($\mathrm{GF}(q^m)$ is thus the smallest extension field of $\mathrm{GF}(q)$ that contains a primitive nth root of unity), and let α be a primitive nth root of unity. Select $g(x)$ to be a minimal-degree polynomial in $\mathrm{GF}(q)[x]$ such that $g(\alpha^b) = g(\alpha^{b+1}) = \cdots = g(\alpha^{b+\delta-2}) = 0$ for some integers $b \geq 0$ and $\delta \geq 1$ (so $g(x)$ has $\delta - 1$ consecutive powers of α as zeros). Now the code C defined by $g(x)$ has minimum distance $d_{\min} \geq \delta$.

The parameter δ in this theorem is the **design distance** of the BCH code defined by $g(x)$.

Parity Check Matrix for BCH Code

The following matrix can be used as a parity check matrix for a BCH code from Theorem 8-1:

$$\begin{bmatrix} 1 & \alpha^{b} & \alpha^{2b} & \cdots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \alpha^{2(b+1)} & \cdots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{b+\delta-3} & \alpha^{2(b+\delta-3)} & \cdots & \alpha^{(n-1)(b+\delta-3)} \\ 1 & \alpha^{b+\delta-2} & \alpha^{2(b+\delta-2)} & \cdots & \alpha^{(n-1)(b+\delta-2)} \end{bmatrix}.$$

(Note: The first column can be written as $\alpha^{0\cdot b}, \alpha^{0\cdot(b+1)}, \ldots$.)

Design Procedure for BCH Codes

To construct a t-error-correcting q-ary BCH codes of length n:

1. Find a primitive nth root of unity $\alpha \in \mathrm{GF}(q^m)$, where m is minimal.
2. Select $\delta - 1 = 2t$ consecutive powers of α, starting with α^b for some nonnegative integer b.
3. Let $g(x)$ be the least common multiple of the minimal polynomials for the selected powers of α with respect to $\mathrm{GF}(q)$. (Each of the minimal polynomials should appear only once in the product.)

Types of BCH Codes

narrow-sense	BCH code with $b = 1$.
primitive	BCH code with $n = q^m - 1$ for some positive integer m (the nth root of unity α is a primitive element in GF(q^m)).

A list of the generator polynomials for binary, narrow-sense, primitive BCH codes of lengths 7 through 255 can be found in [Wic, Appendix E].

Example: Binary BCH Codes of Length 31 (1)

Let α be a root of the primitive polynomial $x^5 + x^2 + 1 \in \mathrm{GF}(2)[x]$. Then α is a primitive element in GF(32), so these BCH codes are *primitive*. The cyclotomic cosets and minimal polynomials are

$$\begin{array}{lcl}
C_0 = \{0\} & \leftrightarrow & M_0(x) = x + 1, \\
C_1 = \{1, 2, 4, 8, 16\} & \leftrightarrow & M_1(x) = x^5 + x^2 + 1, \\
C_3 = \{3, 6, 12, 24, 17\} & \leftrightarrow & M_3(x) = x^5 + x^4 + x^3 + x^2 + 1, \\
C_5 = \{5, 10, 20, 9, 18\} & \leftrightarrow & M_5(x) = x^5 + x^4 + x^2 + x + 1, \\
C_7 = \{7, 14, 28, 25, 19\} & \leftrightarrow & M_7(x) = x^5 + x^3 + x^2 + x + 1, \\
C_{11} = \{11, 22, 13, 26, 21\} & \leftrightarrow & M_{11}(x) = x^5 + x^4 + x^3 + x + 1, \\
C_{15} = \{15, 30, 29, 27, 23\} & \leftrightarrow & M_{15}(x) = x^5 + x^3 + 1.
\end{array}$$

Example: Binary BCH Codes of Length 31 (2)

A narrow-sense one-error-correcting code: Now $b = 1$ and $\delta = 3$, so $g(x)$ must have α and α^2 as zeros. The minimal polynomial of both α and α^2 is $M_1(x)$, so the generator polynomial is

$$g(x) = \mathrm{LCM}(M_1(x), M_2(x)) = M_1(x) = M_2(x) = x^5 + x^2 + 1.$$

Since $\deg(g(x)) = 5$, the dimension of the code is $31 - 5 = 26$, so $g(x)$ defines a $(31, 26)$ binary single-error-correcting BCH code.

Example: Binary BCH Codes of Length 31 (3)

A parity check matrix for the constructed code has the following general form:

$$\mathsf{H} = \begin{bmatrix} 1 & \alpha & \cdots & \alpha^{29} & \alpha^{30} \\ 1 & \alpha^2 & \cdots & \alpha^{27} & \alpha^{29} \end{bmatrix}.$$

Since any binary polynomial having α as a zero must also have the other conjugates as zeros (including α^2), the matrix has redundant rows, so the second row may be deleted.

Note: This code is the binary Hamming code of length 31.

Example: Binary BCH Codes of Length 31 (4)

A narrow-sense two-error-correcting code: Now $b = 1$ and $\delta = 5$, so $g(x)$ must have α, α^2, α^3, and α^4 as zeros. The generator polynomial is

$$\begin{aligned} g(x) &= \mathrm{LCM}(M_1(x), M_2(x), M_3(x), M_4(x)) = M_1(x)M_3(x) \\ &= (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + x + 1) \\ &= x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1. \end{aligned}$$

Since $\deg(g(x)) = 10$, the dimension of the code is $31 - 10 = 21$, so $g(x)$ defines a $(31, 21)$ binary double-error-correcting BCH code.

BCH Codes: Some Remarks

- The true minimum distance may be larger than the design distance.
- We want to maximize the dimension (and therefore the rate) with a given minimum distance. Therefore, it is sometimes worth considering codes that are not narrow-sense ($b > 1$).
- The weight distributions for most BCH codes are not known.
- The weight distributions for all double- and triple-error-correcting binary primitive BCH codes have been found.

Reed-Solomon Codes

Some trends:

1. For a fixed alphabet GF(q), the cardinality of the cyclotomic cosets modulo n is generally smaller for primitive codes ($n = q^m - 1$).
2. Large alphabets generally lead to smaller cyclotomic cosets.

BCH codes of length $n = q - 1$ over GF(q) are called **Reed-Solomon codes**.

Constructing Reed-Solomon Codes

We want to construct a t-error-correcting code of length $q-1$ over GF(q).

1. By Theorem 2-12, there exists a required primitive $(q-1)$th root of unity α in GF(q).

2. We want to construct the cyclotomic cosets modulo $q-1$ with respect to GF(q). Since $q \equiv 1 \pmod{q-1}$, we have $aq^s \equiv a \pmod{q-1}$, so all cyclotomic cosets have one element $\{a\}$ and the associated minimal polynomials are of the form $x - \alpha^a$.

The generator polynomial of a t-error correcting code is then

$$g(x) = (x-\alpha^b)(x-\alpha^{b+1})\cdots(x-\alpha^{b+2t-1}).$$

Example: Reed-Solomon Code over GF(8) (1)

Let α be a root of the primitive binary polynomial $x^3 + x + 1$ and therefore a primitive 7th root of unity. (The elements of GF(8) are then 0, 1, α, α^2, $\alpha^3 = \alpha + 1$, $\alpha^4 = \alpha^2 + \alpha$, $\alpha^5 = \alpha^2 + \alpha + 1$, and $\alpha^6 = \alpha^2 + 1$.)

We construct a 2-error-correcting code. Then $2t = 4$, and a narrow-sense generator polynomial is

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3.$$

Example: Reed-Solomon Code over GF(8) (2)

Since the generator polynomial has degree 4, we have a $(7, 3)$ code over GF(8) and the following parity check matrix:

$$\mathsf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \\ 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 \\ 1 & \alpha^4 & \alpha & \alpha^5 & \alpha^2 & \alpha^6 & \alpha^3 \end{bmatrix}.$$

Minimum Distance of Reed-Solomon Codes

As the following theorem shows, we know the minimum distance of Reed-Solomon codes!

Theorem 8-2. An (n, k) Reed-Solomon code has minimum distance $n - k + 1$.

Proof.
Since the generator polynomial $g(x)$ is the product of $\delta - 1$ minimal polynomials of the form $x - \alpha^a$, its degree is $\delta - 1$. As we also know that the degree of $g(x)$ is $n - k$, we get that the minimum distance is at least $\delta = n - k + 1$. The result now follows, since by the Singleton bound (Theorem 4-10), the minimum distance is at most $n - k + 1$. □

Maximum Distance Separable Codes

An (n, k) code that satisfies the Singleton bound with equality is called **maximum distance separable (MDS)**. MDS codes have a number of interesting properties.

- If C is MDS, so is its dual $C^\perp$.
- Any combination of k coordinates in an MDS code may be used as message coordinates in a systematic representation.
- The weight distribution of MDS codes is known, see [Wic, Theorem 8-5].
- Punctured and shortened MDS codes are MDS.

Decoding BCH and Reed-Solomon Codes

The first explicit decoding algorithm for binary BCH codes was described by Peterson in 1960. Peterson's algorithm is useful only for correcting small numbers of errors.

Berlekamp introduced the first truly efficient decoding algorithm for both binary and nonbinary BCH codes in 1967. This was further developed by Massey and is usually called the *Berlekamp-Massey decoding algorithm*.

These and other decoding algorithms for BCH codes are considered in [Wic, Ch. 9].

Background

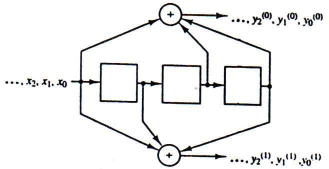
Convolutional codes do not segment the data stream, but convert the entire data stream into another stream (codeword). Some facts about convolutional codes:

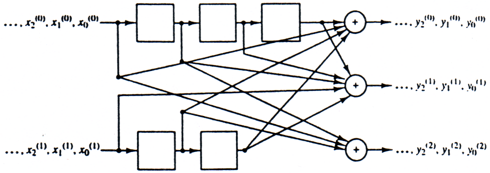
- Introduced by Elias in 1955.
- Constructed mainly using heuristic techniques (cf. block codes: algebraic and combinatorial techniques).
- Can be decoded in an "asymptotically optimal" way using the so-called Viterbi algorithm (which Forney proved to be a maximum-likelihood decoding algorithm for convolutional codes).

Linear Convolutional Encoders (1)

An encoder with k inputs and n outputs is said to have rate k/n.

A rate-1/2 linear convolutional encoder is shown in [Wic, Fig. 11-1], and a rate-2/3 encoder is shown in [Wic, Fig. 11-2]:


$\ldots, x_2, x_1, x_0$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$


$\ldots, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}$
$\ldots, x_2^{(1)}, x_1^{(1)}, x_0^{(1)}$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$
$\ldots, y_2^{(2)}, y_1^{(2)}, y_0^{(2)}$

Linear Convolutional Encoders (2)

The k input and n output streams are denoted by $\mathrm{x}^{(0)}, \mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(k-1)}$, and $\mathrm{y}^{(0)}, \mathrm{y}^{(1)}, \ldots, \mathrm{y}^{(n-1)}$, respectively. The data of an input stream is denoted by $\mathrm{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \ldots)$, and analogously for $\mathrm{y}^{(i)}$.

From multiple output streams, we can create a single output stream $\mathrm{y} = (y_0^{(0)}, y_0^{(1)}, \ldots, y_0^{(n-1)}, y_1^{(0)}, \ldots)$, and an analogous notation can be used for multiple inputs.

Linear Convolutional Encoders (3)

Linear convolutional encoders can be viewed as, for example,

- finite impulse response (FIR) digital filters, or
- finite state automata.

It is known that every linear convolutional encoder is equivalent to a *minimal encoder* that is *feedback-free*.

Output as Function of Input (1)

Each element in the output stream y is a combination of the elements in the input stream x. (Note: The shift-register contents are initialized to zero before the encoding process begins.)

Example. In the encoder in [Wic, Fig. 11-1], we have

$$\begin{aligned}
y_0^{(1)} &= x_0^{(0)} + 0 + 0, \\
y_1^{(1)} &= x_1^{(0)} + x_0^{(0)} + 0, \\
y_2^{(1)} &= x_2^{(0)} + x_1^{(0)} + 0, \\
y_3^{(1)} &= x_3^{(0)} + x_2^{(0)} + x_0^{(0)}, \\
&\vdots \\
y_i^{(1)} &= x_i^{(0)} + x_{i-1}^{(0)} + x_{i-3}^{(0)}.
\end{aligned}$$

Output as Function of Input (2)

Example. (cont.) From the last expression on the previous page, it is clear that if y' and y'' are the codewords corresponding to inputs x' and x'', respectively, then $y' + y''$ is the codeword corresponding to the input $x' + x''$, so the code is *linear*. In this example, if $x^{(0)} = (10110)$, then $y^{(1)} = (11111110)$.

The linear structure of these codes allows for the use of some powerful techniques from linear systems theory.

Impulse Response

An **impulse response** $\mathrm{g}_j^{(i)}$ is obtained for the ith output of an encoder by applying a single 1 at the jth input followed by a string of zeros: $\mathrm{x}^{(j)} = \delta = (1000\ldots)$. Strings of zeros are applied to all other inputs. The impulse response is terminated at the point from which the output contains only zeros.

Example. For the encoder in [Wic, Fig. 11-1], we have

$$\begin{aligned}\mathrm{g}_0^{(0)} &= (1011),\\ \mathrm{g}_0^{(1)} &= (1101).\end{aligned}$$

(Notation: $g_{j,l}^{(i)}$ is bit l of $\mathrm{g}_j^{(i)}$.) The impulse responses are often referred to as **generator sequences**.

Constraint Length

The **constraint length** K of a convolutional code is the maximum number of bits in a single output stream that can be affected by any input bit.

In practice, another definition is often used: The constraint length is the length of the longest input shift register plus one:

$$K := 1 + m,$$

where $m := \max_i m_i$ is called the *maximal memory order* and m_i is the length of the shift register into which $\mathrm{x}^{(i)}$ is fed. The *total memory* for a convolutional encoder is defined as $\sum_{i=0}^{k-1} m_i$.

Fractional Rate Loss

Asymptotically, the ratio between the number of input bits and output bits tends to $R = k/n$. For an input stream of length L, the ratio is not exactly R, but we have a *fractional rate loss*

$$\gamma = \frac{R - R_{\text{eff}}}{R} = \left(\frac{k}{n}\right)^{-1} \left\{ \left(\frac{k}{n}\right) - \left[\frac{L}{\left(\frac{n}{k}\right) L + nm}\right] \right\} = \frac{km}{L + km}.$$

Example. For the encoder in [Wic, Fig. 11-1] we have $k = 1$ and $m = 3$, so with an input of length $L = 5$ (as in an earlier example), we get

$$\gamma = \frac{km}{L + km} = \frac{1 \cdot 3}{5 + 1 \cdot 3} = 0.375.$$

Convolutions of Sequences

An output stream can be expressed as a function of the input streams and the generator sequences

$$y_j^{(i)} = \sum_{t=0}^{k-1} \left(\sum_{l=0}^{m} x_{j-l}^{(t)} g_{t,l}^{(i)} \right),$$

which is a sum of discrete *convolutions* of pairs of sequences:

$$\mathrm{y}^{(i)} = \sum_{t=0}^{k-1} \mathrm{x}^{(t)} * \mathrm{g}_t^{(i)}.$$

Convolutional Generator Matrix

The previous expressions can be re-expressed as a matrix multiplication operation, thus obtaining a semi-infinite generator matrix.

For a rate-1/2 code, the generator matrix is formed from the two generator sequences as follows:

$$\mathrm{G} = \begin{bmatrix} g_0^{(0)} & g_0^{(1)} & g_1^{(0)} & g_1^{(1)} & & \cdots & & g_m^{(0)} & g_m^{(1)} & & \\ & & g_0^{(0)} & g_0^{(1)} & g_1^{(0)} & g_1^{(1)} & & \cdots & & g_m^{(0)} & g_m^{(1)} \\ & & & \ddots & & \ddots & & \ddots & & \ddots & \end{bmatrix}.$$

Example: Convolutional Generator Matrix

The sequence $\mathrm{x} = (1011)$ is to be encoded using the rate-$1/2$ encoder in [Wic, Fig. 11-1]. For this code, $\mathrm{g}^{(0)} = (1011)$ and $\mathrm{g}^{(1)} = (1101)$, so

$$\mathrm{y} = \mathrm{xG} = (1011) \begin{bmatrix} 11 & 01 & 10 & 11 & 00 & 00 & 00 \\ 00 & 11 & 01 & 10 & 11 & 00 & 00 \\ 00 & 00 & 11 & 01 & 10 & 11 & 00 \\ 00 & 00 & 00 & 11 & 01 & 10 & 11 \end{bmatrix} =$$

$(11, 01, 01, 01, 11, 01, 11)$.

The Delay Transform (1)

An appropriate transform will provide a simpler multiplicative representation for encoding. In this case, we apply the **delay transform** (also called the *D*-transform):

$$\begin{aligned}
\mathrm{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \ldots) \quad &\leftrightarrow \quad \mathrm{X}^{(i)}(D) = x_0^{(i)} + x_1^{(i)} D + \cdots , \\
\mathrm{y}^{(i)} = (y_0^{(i)}, y_1^{(i)}, \ldots) \quad &\leftrightarrow \quad \mathrm{Y}^{(i)}(D) = y_0^{(i)} + y_1^{(i)} D + \cdots , \\
\mathrm{g}_j^{(i)} = (g_{j,0}^{(i)}, g_{j,1}^{(i)}, \ldots) \quad &\leftrightarrow \quad \mathrm{G}_j^{(i)}(D) = g_{j,0}^{(i)} + g_{j,1}^{(i)} D + \cdots .
\end{aligned}$$

The Delay Transform (2)

The encoding operation can now be represented as

$$\mathrm{Y}^{(i)}(D) = \sum_{j=0}^{k-1} \mathrm{X}^{(j)}(D)\mathrm{G}_j^{(i)}(D),$$

or as (the matrix $\mathrm{G}(D)$ is called a *transfer-function matrix*)

$$\mathrm{Y}(D) = \mathrm{X}(D)\mathrm{G}(D) = (\mathrm{X}^{(0)}(D)\ \mathrm{X}^{(1)}(D)\ \cdots\ \mathrm{X}^{(k-1)}(D))\cdot$$

$$\begin{bmatrix} \mathrm{G}_0^{(0)}(D) & \mathrm{G}_0^{(1)}(D) & \cdots & \mathrm{G}_0^{(n-1)}(D) \\ \mathrm{G}_1^{(0)}(D) & \mathrm{G}_1^{(1)}(D) & \cdots & \mathrm{G}_1^{(n-1)}(D) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{G}_{k-1}^{(0)}(D) & \mathrm{G}_{k-1}^{(1)}(D) & \cdots & \mathrm{G}_{k-1}^{(n-1)}(D) \end{bmatrix}.$$

Example: The Delay Transform

The rate-2/3 encoder in [Wic, Fig. 11-2] is used to encode the message $x = (11, 10, 11)$. Then $\mathrm{X}^{(0)}(D) = 1 + D + D^2$ and $\mathrm{X}^{(1)}(D) = 1 + D^2$, so

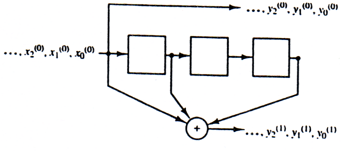
$$\mathrm{Y}(D) = (1 + D + D^2 \;\; 1 + D^2) \left[\begin{array}{ccc} 1 + D^3 & D + D^2 + D^3 & 1 + D \\ D + D^2 & 1 + D^2 & D \end{array} \right] =$$

$$(1 + D^5 \;\; 1 + D + D^3 + D^4 + D^5 \;\; 1 + D).$$

Hence $\mathrm{y}^{(0)} = (100001)$, $\mathrm{y}^{(1)} = (110111)$, and $\mathrm{y}^{(2)} = (110000)$, and the output word is $\mathrm{y} = (111, 011, 000, 010, 010, 110)$.

Systematic Convolutional Codes

A convolutional code is said to be **systematic** if the input data is reproduced unaltered in the output codeword.


$\ldots, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$+$
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$

A rate-$1/2$ systematic convolutional encoder is shown in [Wic, Fig. 11-3]. In a rate-k/n systematic encoder, k of the n output coded data streams are identical to the k input streams. The matrix $\mathsf{G}(D)$ of a systematic encoder contains the $k \times k$ identity matrix I_k.

Note: Not every convolutional code has a systematic encoding (which is the case for block codes). In fact, many good convolutional codes are not systematic.

Properties of Convolutional Codes

Convolutional codes differ from block codes in several ways, some mentioned earlier. Most of the techniques used for analyzing and comparing block codes cannot be applied to convolutional codes. For example, what about minimum distance?

There are two important graphical techniques for analyzing convolutional codes:

1. State diagrams.
2. Trellis diagrams.

State Diagrams (1)

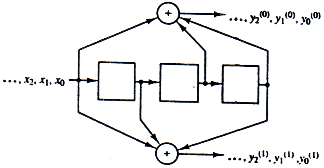
A convolutional encoder is a finite-state automaton, where the next output depends only on the input bits and the contents of its memory cells. With k memory cells, the finite-state automaton has 2^k states.

Example. The encoder in [Wic, Fig. 11-1] has 8 states, which can be associated with the memory contents as follows:

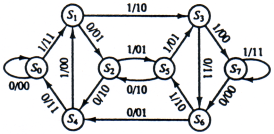
$$\begin{array}{ll} S_0 \leftrightarrow (000), & S_4 \leftrightarrow (001), \\ S_1 \leftrightarrow (100), & S_5 \leftrightarrow (101), \\ S_2 \leftrightarrow (010), & S_6 \leftrightarrow (011), \\ S_3 \leftrightarrow (110), & S_7 \leftrightarrow (111). \end{array}$$

State Diagrams (2)

Example. (cont.) The **state diagram** of this encoder is shown in [Wic, Fig. 11-4]. Each branch in the state diagram has a label of the for X/YY, where X is the input bit that causes the state transition and YY is the corresponding pair of output bits.



..., $y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
..., x_2, x_1, x_0
..., $y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$



Some Graph Concepts

graph	Consists of vertices (nodes) and edges (branches) that connect the vertices.
directed graph	A graph where the edges have an associated direction.
weighted graph	A graph where (usually nonnegative integer) values are associated with the edges.
path	A sequence of vertices in which consecutive vertices are connected with an edge (in the correct direction if the graph is directed).
circuit	A path that starts and stops at the same vertex.
cycle	A circuit in which the only repeated vertex is the first one.

State Diagrams (3)

We may consider the state diagram as a *weighted directed graph*, where the weight of an edge is the Hamming weight of the output bits. State diagrams are therefore also called *encoder graphs*.

The encoding process begins and ends in the all-zero state, so every convolutional codeword is associated with a circuit through the encoder graph that starts and stops at state S_0.

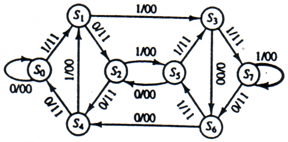
Catastrophic Convolutional Codes

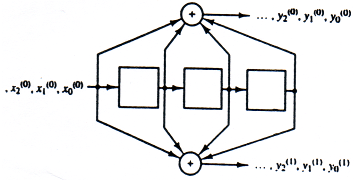
A convolutional code is said to be **catastrophic** if its corresponding state diagram contains a cycle in which a nonzero input sequence corresponds to an all-zero output sequence.

With a catastrophic code, a small number of channel errors can cause an unlimited number of errors in the decoded data stream.

Example: Catastrophic Code (1)

The encoder graph of a catastrophic code is shown in [Wic, Fig. 11-5] (and the corresponding encoder is depicted in [Wic, Fig. 11-6]):




$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}$
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$
+
+

Example: Catastrophic Code (2)

The loop about state S_7 shows that the code is catastrophic. With this encoder, an all-zero word $(0, 0, \ldots, 0)$ is encoded as

$$(00, 00, \ldots, 00),$$

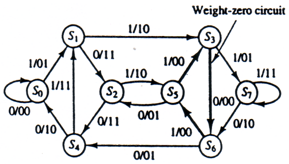
and an all-one word $(1, 1, \ldots, 1)$ is encoded as

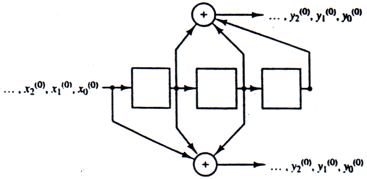
$$(11, 00, 11, 00, 00, \ldots, 00, 00, 11, 00, 11).$$

Then there are situations with a small number of channel errors where maximum-likelihood decoding will lead to a huge number of errors.

Avoiding Catastrophic Codes (1)

To avoid the weight-zero loop about the all-ones state, one need only assure that at least one of the output streams is formed through the summation of an odd number of terms. This condition is not sufficient, however, as shown by the encoder and the encoder graph given in [Wic, Fig. 11-8] and [Wic, Fig. 11-7], respectively:




$\ldots, x_2^{(0)}, x_1^{(0)}, x_0^{(0)}$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$

Avoiding Catastrophic Codes (2)

Fortunately, catastrophic codes are relatively infrequent (for example, only $1/(2^n - 1)$ of all convolutional codes of rate $1/n$ and a given constraint length are catastrophic).

A set of necessary and sufficient conditions for a convolutional code to be noncatastrophic has been proved by Massey and Sain.

Conditions for Catastrophic Codes

1. Let C be a rate-$1/n$ convolutional code with transfer-function matrix $\mathrm{G}(D)$ whose generator sequences have the transforms $\mathrm{G}^{(i)}(D)$, $0 \le i \le n-1$. Then C is not catastrophic iff for some nonnegative integer j,

$$\mathrm{GCD}(\mathrm{G}^{(0)}(D), \mathrm{G}^{(1)}(D), \ldots, \mathrm{G}^{(n-1)}(D)) = D^j.$$

2. Let C be a rate-k/n convolutional code with transfer-function matrix $\mathrm{G}(D)$. Then C is not catastrophic iff for some nonnegative integer j,

$$GCD(\Delta_i(D) \mid i \in I) = D^j,$$

where $\Delta_i(D)$ is the determinant of the ith $k \times k$ submatrix of $\mathrm{G}(D)$ and I contains the indexes of all such submatrices.

Example: Conditions for Convolutional Codes

The generator sequences for the encoder in [Wic, Fig. 11-8] are

$$\mathrm{g}^{(0)} = (0111),$$
$$\mathrm{g}^{(1)} = (1110),$$

and the corresponding D-transforms are

$$\mathsf{G}^{(0)} = D + D^2 + D^3,$$
$$\mathsf{G}^{(1)} = 1 + D + D^2.$$

Then $\mathsf{GCD}(\mathsf{G}^{(0)}, \mathsf{G}^{(1)}) = 1 + D + D^2$, and since $1 + D + D^2 \neq D^j$ for any integer j, the code is catastrophic.

Performance Measures for Convolutional Codes

Whereas there is one main performance measure for block codes, minimum distance, there are several possible performance measures for convolutional codes. The following three are considered here:

1. The column distance function.
2. Minimum distance.
3. Minimum free distance.

The Column Distance Function (1)

Consider a rate-k/n code C with constraint length K. Let the input sequence x and the output sequence y for an encoder for C be truncated at length i:

$$[\mathrm{x}]_i = (x_0^{(0)}, \dots, x_0^{(k-1)}, x_1^{(0)}, \dots, x_1^{(k-1)}, \dots, x_{i-1}^{(0)}, \dots, x_{i-1}^{(k-1)}),$$
$$[\mathrm{y}]_i = (y_0^{(0)}, \dots, y_0^{(n-1)}, y_1^{(0)}, \dots, y_1^{(n-1)}, \dots, y_{i-1}^{(0)}, \dots, y_{i-1}^{(n-1)}).$$

The **column distance function** (CDF) d_i is the minimum Hamming distance between all pairs of output sequences truncated at length i given that the input sequences differ in the first k bits:

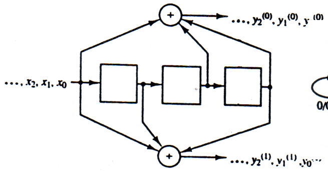
$$d_i := \min\{d([\mathrm{y}']_i, [\mathrm{y}'']_i) \mid [\mathrm{x}']_1 \neq [\mathrm{x}'']_1\}.$$

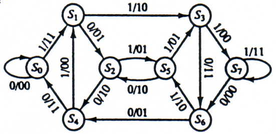
The Column Distance Function (2)

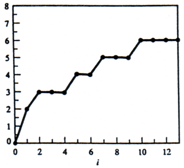
If the convolutional code is linear, the CDF can be defined in the following way:

$$d_i := \min\{w([\mathrm{y}]_i) \mid [\mathrm{x}]_1 \neq 0\}.$$

Example. The CDF for the code in [Wic, Fig. 11-1] is shown in [Wic, Fig. 11-13].


+
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$\ldots, x_2, x_1, x_0$
+
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$





Minimum Distance

The **minimum distance** d_{min} of a rate-k/n convolutional code with constraint length K is d_K (that is, the CDF evaluated at $i = K$).

Example. The minimum distance for the code defined by the decoder in [Wic, Fig. 11-1] is $d_{\text{min}} = d_4 = 3$.

The parameter d_{min} is useful for methods that use nK bits of a received word to decode a single bit.

Minimum Free Distance (1)

The **minimum free distance** d_{free} is the minimum Hamming distance between all pairs of complete convolutional codewords,

$$\begin{aligned} d_{\text{free}} &:= \min\{d(\mathrm{y}', \mathrm{y}'') \mid \mathrm{y}' \neq \mathrm{y}''\} \\ &= \min\{w(\mathrm{y}) \mid \mathrm{y} \neq 0\}. \end{aligned}$$

The minimum free distance is important in particular for the Viterbi decoder (to be considered), which uses the entire codeword to decode a single bit. It can be obtained by finding a cycle of minimum weight in the encoder graph starting and stopping at S_0.

Minimum Free Distance (2)

Example. The minimum free distance for the code defined by the decoder in [Wic, Fig. 11-1] is $d_{\text{free}} = 6$.

This and a previous example show the expected result that better performance is provided by techniques that use the entire word instead of nK bits to decode a single bit ($d_{\text{free}} = 6 > d_{\min} = 3$).

Some Results on Distance Parameters

- ▷ For noncatastrophic codes, $\lim_{i \to \infty} d_i = d_{\text{free}}$.
- ▷ Unlike linear block codes, *nonsystematic* convolutional codes often offer a higher minimum free distance than systematic codes of comparable constraint length and rate (cf. [Wic, Tables 11-1 and 11-2]).

Lists of nonsystematic rate-1/4, rate-1/3, rate-1/2, rate-2/3, and rate-3/4 convolutional codes with largest possible minimum free distance for small constraint lengths are listed in [Wic, Tables 11-3 to 11-7].

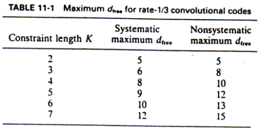


TABLE 11-1 Maximum d_{free} for rate-1/3 convolutional codes

Constraint length K	Systematic maximum d_{free}	Nonsystematic maximum d_{free}
2	5	5
3	6	8
4	8	10
5	9	12
6	10	13
7	12	15

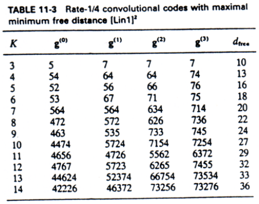


TABLE 11-3 Rate-1/4 convolutional codes with maximal minimum free distance [Lin1]^2

K	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbf{g}^{(2)}$	$\mathbf{g}^{(3)}$	d_{free}
3	5	7	7	7	10
4	54	64	64	74	13
5	52	56	66	76	16
6	53	67	71	75	18
7	564	564	634	714	20
8	472	572	626	736	22
9	463	535	733	745	24
10	4474	5724	7154	7254	27
11	4656	4726	5562	6372	29
12	4767	5723	6265	7455	32
13	44624	52374	66754	73534	33
14	42226	46372	73256	73276	36

The Viterbi Decoding Algorithm

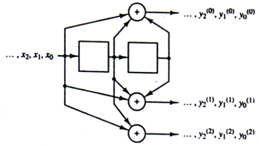
- In 1967, Viterbi proposed an algorithm for “asymptotically optimal” decoding of convolutional codes in memoryless noise.
- It turned out that the algorithm had been invented in operations research ten years earlier, when Minty presented an algorithm for a shortest-route problem.
- The Viterbi algorithm provides both a maximum-likelihood (ML) and a maximum a posteriori (MAP) decoding algorithm for convolutional codes.

The ML version of the algorithm will be considered here.

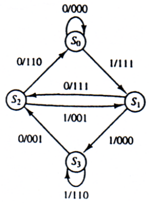
Trellis Diagrams

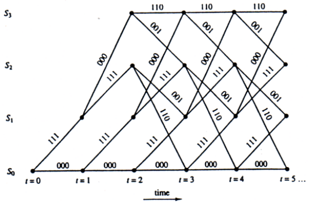
A **trellis diagram** is an extension of the state diagram of a convolutional code, which explicitly shows the passage of time.

Example. A rate-1/3 encoder with two memory cells is shown in [Wic, Fig. 12-1] and its associated state diagram (which has $2^2 = 4$ states) in [Wic, Fig. 12-2]. In [Wic, Fig. 12-3], the state diagram is extended in time to form a trellis diagram. The edges of the trellis diagram are labeled with the corresponding output bits.



$\ldots, x_2, x_1, x_0$
$\ldots, y_2^{(0)}, y_1^{(0)}, y_0^{(0)}$
$\ldots, y_2^{(1)}, y_1^{(1)}, y_0^{(1)}$
$\ldots, y_2^{(2)}, y_1^{(2)}, y_0^{(2)}$



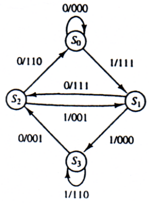


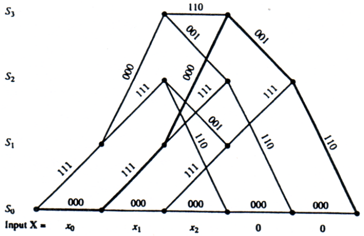
Some Properties of Trellis Diagrams (1)

- ▷ Every codeword in a convolutional code is associated with a unique path through the trellis diagram, starting and stopping at S_0.
- ▷ With total memory M and maximal memory order m, the trellis diagram has 2^M vertices at each time increment after time $t = m$.
- ▷ There are 2^k edges leaving each vertex. After time $t = m$, there are 2^k edges entering each vertex.
- ▷ Given an input sequence of kL bits, the trellis diagram has $L + m$ stages, and codewords have $n(L + m)$ bits.
- ▷ There are 2^{kL} different paths through a trellis diagram with $L + m$ stages.

Some Properties of Trellis Diagrams (2)

Example. The input sequence $x = (011)$ to the encoder in [Wic, Fig. 12-1] is shown in [Wic, Fig. 12-4]. Since $L = 3$, $n = 3$, and $m = 2$, the codeword has $n(L + m) = 3(3 + 2) = 15$ bits and is $y = (000, 111, 000, 001, 110)$.





The Viterbi Algorithm (1)

In a communication system, an information stream x is encoded into a convolutional codeword y, which is transmitted across a (noisy) channel. At the receiving end, we want to find a good estimate y' of the transmitted word given the received word r. Analogously to an earlier discussion for block codes,

- ▷ a maximum a posteriori (MAP) decoder selects an estimate y' that maximizes $p(\mathrm{y}' \mid \mathrm{r})$,
- ▷ a maximum likelihood (ML) decoder selects an estimate y' that maximizes $p(\mathrm{r} \mid \mathrm{y}')$, and
- ▷ these two decoders are identical when the distribution of the source words $\{\mathrm{x}\}$ is uniform.

The Viterbi Algorithm (2)

With a rate-k/n convolutional encoder and an input sequence x composed of L k-bit blocks

$$\mathrm{x} = (x_0^{(0)}, x_0^{(1)}, \ldots, x_0^{(k-1)}, x_1^{(0)}, x_1^{(1)}, \ldots, x_1^{(k-1)}, \ldots, x_{L-1}^{(k-1)}),$$

the output sequence y will consist of $L+m$ n-bit blocks (where m is the maximal memory order)

$$\mathrm{y} = (y_0^{(0)}, y_0^{(1)}, \ldots, y_0^{(n-1)}, y_1^{(0)}, y_1^{(1)}, \ldots, y_1^{(n-1)}, \ldots, y_{L+m-1}^{(n-1)}).$$

The corrupted word r that arrives at the receiver is

$$\mathrm{r} = (r_0^{(0)}, r_0^{(1)}, \ldots, r_0^{(n-1)}, r_1^{(0)}, r_1^{(1)}, \ldots, r_1^{(n-1)}, \ldots, r_{L+m-1}^{(n-1)})$$

The Viterbi Algorithm (3)

The decoder generates a maximum likelihood estimate

$$\mathrm{y}' = (y_0'^{(0)}, y_0'^{(1)}, \ldots, y_0'^{(n-1)}, y_1'^{(0)}, y_1'^{(1)}, \ldots, y_1'^{(n-1)}, \ldots, y_{L+m-1}'^{(n-1)}).$$

We assume that the channel is **memoryless**, that is, that the noise process affecting a given bit is independent of the noise process affecting any other bits. Then

$$p(\mathrm{r} \mid \mathrm{y}') = \prod_{i=0}^{L+m-1} \left(\prod_{j=0}^{n-1} p(r_i^{(j)} \mid y_i'^{(j)}) \right).$$

The Viterbi Algorithm (4)

For easier calculations, we take the logarithm of the previous expression (note that logarithms are monotonically increasing and $\log xy = \log x + \log y$) and transform it $x \to a(x+b)$ to get the **path metric**

$$M(\mathrm{r} \mid \mathrm{y}') = \sum_{i=0}^{L+m-1} \left(\sum_{j=0}^{n-1} M(r_i^{(j)} \mid y_i'^{(j)}) \right),$$

where

$$M(r_i^{(j)} \mid y_i'^{(j)}) = a[\log p(r_i^{(j)} \mid y_i'^{(j)}) + b].$$

The Viterbi Algorithm (5)

The *s*th **partial path metric** is defined as

$$M^s(\mathrm{r} \mid \mathrm{y}') = \sum_{i=0}^{s} \left(\sum_{j=0}^{n-1} M(r_i^{(j)} \mid y_i'^{(j)}) \right).$$

Until we reach the point in the trellis where more than one path enters each node, we assign to a node the value $M^s(\mathrm{r} \mid \mathrm{y}')$ of the only possible path. From that point on, the optimal partial path metric among the metrics for all of the entering paths is chosen. This is shown in [Wic, p. 296].

The Complete Viterbi Algorithm (1)

Let the vertex corresponding to state S_j at time t be denoted $S_{j,t}$. Each such vertex is assigned a value $V(S_{j,t})$ in the following way.

1. Set $V(S_{0,0}) = 0$ and $t = 1$.

2. At time t, compute the partial path metrics for all paths entering each vertex.

3. Set $V(S_{k,t})$ equal to the best partial path metric entering a vertex corresponding to state S_k at time t. One best edge survives (ties are randomly broken); the others are deleted from the trellis.

4. If $t < L + m$, $t := t + 1$ and goto Step 2; otherwise Stop.

The Complete Viterbi Algorithm (2)

Once all vertex values have been computed, the vector y′ is obtained by starting at state S_0, time $t = L + m$ and following the surviving edges backward through the trellis.

Theorem 12-1. The path selected by the Viterbi decoder is the maximum likelihood path.

Hard-Decision Decoding

In **hard-decision decoding** each received bit is examined and a decision is made whether it represents a 0 or a 1. A **binary memoryless channel** model is depicted in [Wic, Fig. 12-6]. If $p(0 \mid 1) = p(1 \mid 0)$, we have a **binary symmetric channel (BSC)**.

The Viterbi Algorithm for the BSC (1)

For the BSC we have $p(0 \mid 1) = p(1 \mid 0) = p$ and $p(0 \mid 0) = p(1 \mid 1) = 1 - p$. By letting $a = (\log_2 p - \log_2(1-p))^{-1}$ (which is negative when $p < 1/2$) and $b = -\log_2(1-p)$,

$$M(r_i^{(j)} \mid y_i^{(j)}) = \frac{1}{\log_2 p - \log_2(1-p)}[\log_2 p(r_i^{(j)} \mid y_i^{(j)}) - \log_2(1-p)],$$

and we get the following table.

$M(r_i^{(j)} \mid y_i^{(j)})$	$r_i^{(j)} = 0$	$r_i^{(j)} = 1$
$y_i^{(j)} = 0$	0	1
$y_i^{(j)} = 1$	1	0

The Viterbi Algorithm for the BSC (2)

Since $a < 0$, we have a minimizing problem. For the BSC case, the path metric is simply the Hamming distance!

To get a maximizing problem, we can instead choose $a = (\log_2(1-p) - \log_2 p)$ and $b = -\log_2 p$; then we get the following metric.

$M(r_i^{(j)} \mid y_i^{(j)})$	$r_i^{(j)} = 0$	$r_i^{(j)} = 1$
$y_i^{(j)} = 0$	1	0
$y_i^{(j)} = 1$	0	1

Example: Viterbi Decoding for the BSC (1)

When the encoder in [Wic, Fig. 12-1] is used, the sequence $\mathrm{x} = (110101)$ results in the codeword

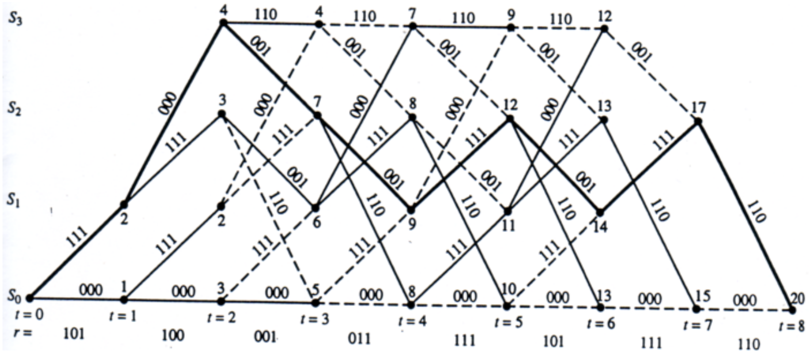
$$\mathrm{y} = (111, 000, 001, 001, 111, 001, 111, 110).$$

Assume that when the word y is transmitted over a noisy channel, the following word is received:

$$\mathrm{r} = (1\bar{0}1, \bar{1}00, 001, 0\bar{1}1, 111, \bar{1}01, 111, 110).$$

In the expression of r, a bar over a bit indicates an error. Using the second set of bit metrics for the BSC case, the result of the decoding is shown in [Wic, Fig. 12-8]. Several ties occur during decoding (for example, for S_3 at times $t = 3$ and $t = 5$); however, these have no impact on the path selected by the decoder.

Example: Viterbi Decoding for the BSC (2)



Soft-Decision Decoding (1)

In **soft-decision decoding** the receiver takes advantage of side information in a received bit. Instead of assigning a 0 or a 1, as in hard-decision decoding, a more flexible approach is taken through the use of multibit quantization (cf. fuzzy logic!).

Four or more decision regions are established, ranging from a *strong one* to a *strong zero*.

Soft-Decision Decoding (2)

Let the transmitted bits $\{y_i^{(j)}\}$ take the values ± 1. The path metric for the AWGN channel are then the inner product of the received word and the codeword. The individual bit metrics are

$$M(r_i^{(j)} \mid y_i^{(j)}) = r_i^{(j)} y_i^{(j)}.$$

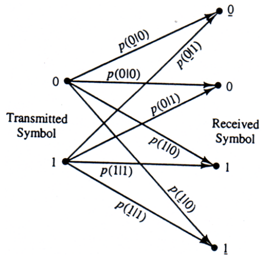
Maximization of the path metric is equivalent to finding the codeword y that is closest to r in terms of Euclidean distance (note: for the BSC, Hamming distance was considered).

In dealing with Euclidean distance and real numbers, finite precision of digital computers will have an impact on the result. In practice, however, this impact turns out to be negligible.

Discrete Symmetric Channels

The fact that the received signal can take more than two values is modeled as a **discrete symmetric channel**. Such a channel with four values for the received signal is depicted in [Wic, Fig. 12-10]. The four values are

- $\underline{0}$ strong 0,
- 0 weak 0,
- 1 weak 1, and
- $\underline{1}$ strong 1.



Example: Soft-Decision Viterbi Decoding (1)

Decoding is performed almost exactly as in the hard-decision case, the only difference being the increased number (and resolution) of the bit metrics.

We consider the following conditional probabilities in [Wic, Fig. 12-10].

$p(r \mid y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	0.50	0.32	0.13	0.05
$y = 1$	0.05	0.13	0.32	0.50

Example: Soft-Decision Viterbi Decoding (2)

$\log_2 p(r \mid y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	-1.00	-1.64	-2.94	-4.32
$y = 1$	-4.32	-2.94	-1.64	-1.00

With $M(r \mid y) = 1.5[\log_2 p(r \mid y) - \log_2(0.05)]$, we get the following table.

$M(r \mid y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	5	4	2	0
$y = 1$	0	2	4	5

Example: Soft-Decision Viterbi Decoding (3)

As in a previous example, assume that the word

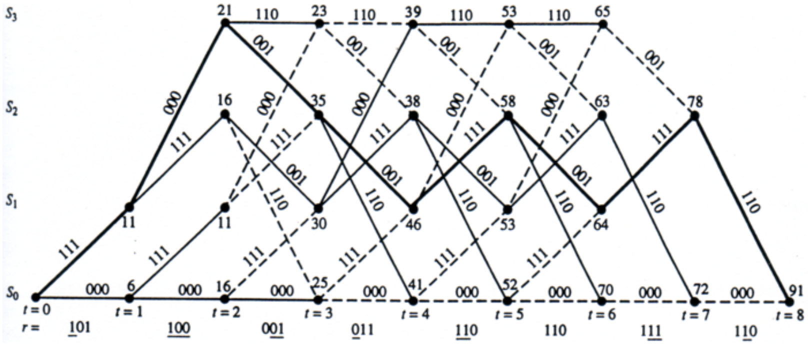
$$\mathrm{y} = (111, 000, 001, 001, 111, 001, 111, 110)$$

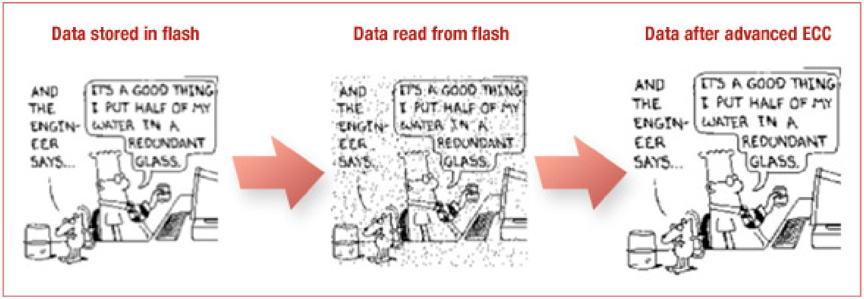
is transmitted, and that the received word is

$$\mathrm{r} = (\underline{1}\overline{0}1, \underline{\overline{1}00}, 0\underline{01}, \underline{0}\overline{1}1, \underline{11}\overline{0}, \overline{110}, 1\underline{11}, 1\underline{1}0).$$

The soft-decoding trellis diagram and the result of the decoding are shown in [Wic, Fig. 12-11].

Example: Soft-Decision Viterbi Decoding (4)




Data stored in flash
Data read from flash
Data after advanced ECC
AND THE ENGINEER SAYS...
IT'S A GOOD THING I PUT HALF OF MY WATER IN A REDUNDANT GLASS.

Background

The material in the sequel is based on

[MF] J. C. Moreira and P. G. Farrell, Essentials of Error-Control Coding, Wiley, Chichester, 2006, pp. 209–325.

- Traditional paradigm: Try to construct codes with large minimum distance.
- Shannon: Random codes are good.
- Modern paradigm: "Distance isn't everything". Two code types with good practical performance are *turbo* and *LDPC* codes. Iterative decoding.

Background

The material in the sequel is based on

[MF] J. C. Moreira and P. G. Farrell, Essentials of Error-Control Coding, Wiley, Chichester, 2006, pp. 209–325.

- Traditional paradigm: Try to construct codes with large minimum distance.
- Shannon: Random codes are good.
- Modern paradigm: "Distance isn't everything". Two code types with good practical performance are *turbo* and *LDPC* codes. Iterative decoding.

Background

The material in the sequel is based on

[MF] J. C. Moreira and P. G. Farrell, Essentials of Error-Control Coding, Wiley, Chichester, 2006, pp. 209–325.

- Traditional paradigm: Try to construct codes with large minimum distance.
- Shannon: Random codes are good.
- Modern paradigm: "Distance isn't everything". Two code types with good practical performance are *turbo* and *LDPC* codes. Iterative decoding.

Background

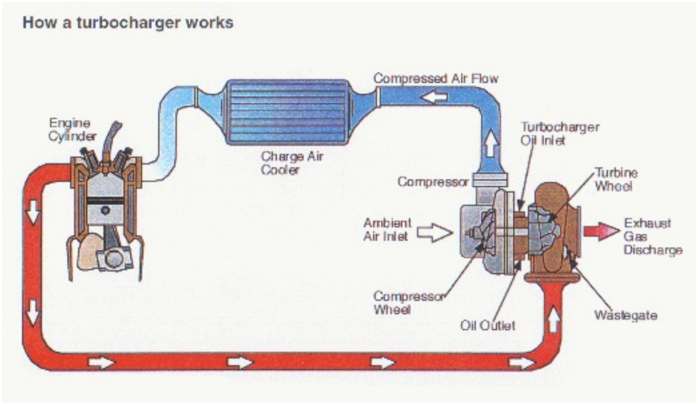
The material in the sequel is based on

[MF] J. C. Moreira and P. G. Farrell, Essentials of Error-Control Coding, Wiley, Chichester, 2006, pp. 209–325.

- Traditional paradigm: Try to construct codes with large minimum distance.
- Shannon: Random codes are good.
- Modern paradigm: "Distance isn't everything". Two code types with good practical performance are *turbo* and *LDPC* codes. Iterative decoding.

Turbo Engines

Turbo(charger): A supercharger driven by a turbine powered by the engine's exhaust gases.



Turbo Codes

The term *turbo* in the coding context origins from some similarity between decoding and the turbo engine principle (and not from the fact that turbo can be imagined as something better and/or faster, although these codes indeed have good practical performance).

Turbo codes were discovered in 1993 by Berrou, Glavieux and Thitimajshima.

Turbo Encoder

A basic turbo encoder is the following rate-$1/3$ systematic encoder:

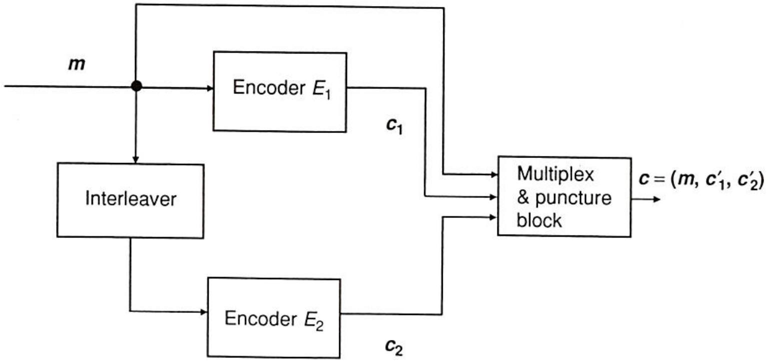


Figure 7.1 A turbo encoder

Turbo Encoder (cont.)

The shown turbo encoder consists of three parallel parts whose output is

1. the data stream unaltered,
2. data from a convolutional encoder,
3. data from a convolutional encoder combined with a random interleaver.

Puncturing (which is usually not applied to the systematic bits) can be used to obtain higher rates than 1/3.

Turbo Encoder (cont.)

The shown turbo encoder consists of three parallel parts whose output is

1. the data stream unaltered,
2. data from a convolutional encoder,
3. data from a convolutional encoder combined with a random interleaver.

Puncturing (which is usually not applied to the systematic bits) can be used to obtain higher rates than 1/3.

Turbo Encoder (cont.)

The shown turbo encoder consists of three parallel parts whose output is

1. the data stream unaltered,
2. data from a convolutional encoder,
3. data from a convolutional encoder combined with a random interleaver.

Puncturing (which is usually not applied to the systematic bits) can be used to obtain higher rates than 1/3.

Turbo Encoder (cont.)

The shown turbo encoder consists of three parallel parts whose output is

1. the data stream unaltered,
2. data from a convolutional encoder,
3. data from a convolutional encoder combined with a random interleaver.

Puncturing (which is usually not applied to the systematic bits) can be used to obtain higher rates than $1/3$.

Turbo Encoder (cont.)

The shown turbo encoder consists of three parallel parts whose output is

1. the data stream unaltered,
2. data from a convolutional encoder,
3. data from a convolutional encoder combined with a random interleaver.

Puncturing (which is usually not applied to the systematic bits) can be used to obtain higher rates than $1/3$.

Turbo Decoder

Turbo codes are decoded in an iterative soft-decision decoding process. Two different decoders—corresponding to the two different encoders—get progressively better estimates of the message bits through an iterative exchange of information.

More precisely, each of the decoders obtain a new estimate based on

- the systematic bits,
- the parity bits (produced by the corresponding encoder),
- the previous estimate of the other decoder.

Turbo Decoder

Turbo codes are decoded in an iterative soft-decision decoding process. Two different decoders—corresponding to the two different encoders—get progressively better estimates of the message bits through an iterative exchange of information.

More precisely, each of the decoders obtain a new estimate based on

- the systematic bits,
- the parity bits (produced by the corresponding encoder),
- the previous estimate of the other decoder.

Turbo Decoder

Turbo codes are decoded in an iterative soft-decision decoding process. Two different decoders—corresponding to the two different encoders—get progressively better estimates of the message bits through an iterative exchange of information.

More precisely, each of the decoders obtain a new estimate based on

- the systematic bits,
- the parity bits (produced by the corresponding encoder),
- the previous estimate of the other decoder.

Turbo Decoder

Turbo codes are decoded in an iterative soft-decision decoding process. Two different decoders—corresponding to the two different encoders—get progressively better estimates of the message bits through an iterative exchange of information.

More precisely, each of the decoders obtain a new estimate based on

- the systematic bits,
- the parity bits (produced by the corresponding encoder),
- the previous estimate of the other decoder.

Turbo Decoder (cont.)

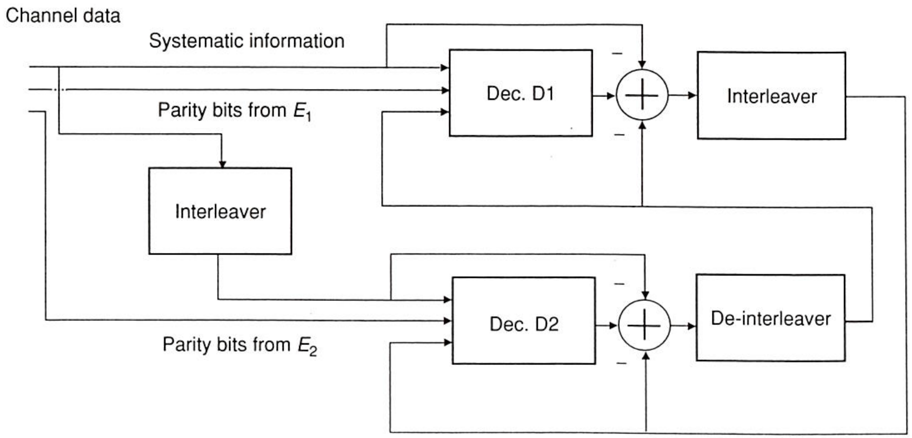


Figure 7.2 A turbo decoder

Log Likelihood Ratio

The estimates are usually communicated from one decoder to the other in the form of *log likelihood ratio* (LLR).

Instead of elements from $\{0, 1\}$, it is often more convenient to consider elements from $\{-1, 1\}$. The LLR is defined as

$$L(b_i) = \ln\left(\frac{P(b_i = +1)}{P(b_i = -1)}\right).$$

The sign of the LLR can be used as the hard decision of the estimate; the absolute value gives the reliability of the estimate.

BCJR Algorithm

The *BCJR algorithm* (Bahl-Cocke-Jelinek-Raviv) determines an estimate for a given sequence element by observing the output sequence of a discrete memoryless channel that corresponds to a given input sequence.

The BCJR algorithm is the core part of iterative decoding of turbo codes. We omit the details here.

Iterative Decoding

We shall next show one scheme for iterative decoding of turbo codes.

A priori information: Information provided by the other encoder. (In the beginning $P(b_i = +1) = P(b_i = -1) = 0.5$ so the a priori LLR is 0 for the first encoder in the first iteration.)

Extrinsic information: The component of the generated reliability value that depends on redundant information introduced by the considered constituent code.

Scheme for Iterative Decoding

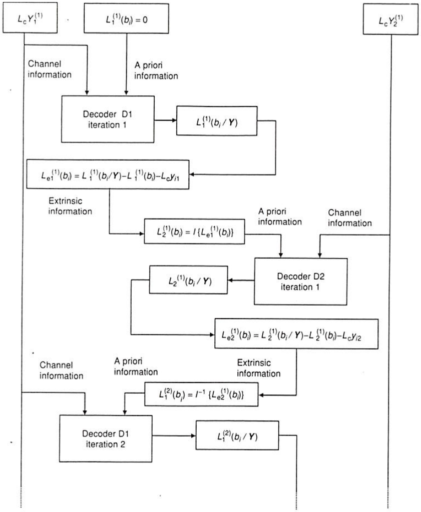

$L_cY_1^{(1)}$
$L_1^{(1)}(b_i)=0$
$L_cY_2^{(1)}$
Channel information
A priori information
Decoder D1 iteration 1
$L_1^{(1)}(b_i/\mathbf{Y})$
$L_{e1}^{(1)}(b_i)=L_1^{(1)}(b_i/\mathbf{Y})-L_1^{(1)}(b_i)-L_cy_{i1}$
Extrinsic information
$L_2^{(1)}(b_i)=I\{L_{e1}^{(1)}(b_i)\}$
A priori information
Channel information
Decoder D2 iteration 1
$L_2^{(1)}(b_i/\mathbf{Y})$
$L_{e2}^{(1)}(b_i)=L_2^{(1)}(b_i/\mathbf{Y})-L_2^{(1)}(b_i)-L_cy_{i2}$
Channel information
A priori information
Extrinsic information
$L_1^{(2)}(b_i)=I^{-1}\{L_{e2}^{(1)}(b_i)\}$
Decoder D1 iteration 2
$L_1^{(2)}(b_i/\mathbf{Y})$

Figure 7.11 Iterative decoding of turbo codes

Example: Iterative Decoding

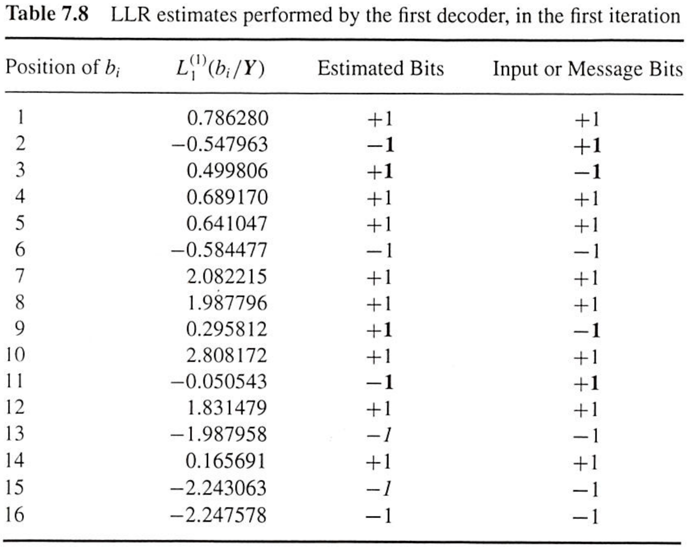


Table 7.8 LLR estimates performed by the first decoder, in the first iteration

Position of b_i	$L_1^{(1)}(b_i/Y)$	Estimated Bits	Input or Message Bits
1	0.786280	+1	+1
2	−0.547963	**−1**	**+1**
3	0.499806	**+1**	**−1**
4	0.689170	+1	+1
5	0.641047	+1	+1
6	−0.584477	−1	−1
7	2.082215	+1	+1
8	1.987796	+1	+1
9	0.295812	**+1**	**−1**
10	2.808172	+1	+1
11	−0.050543	**−1**	**+1**
12	1.831479	+1	+1
13	−1.987958	*−1*	−1
14	0.165691	+1	+1
15	−2.243063	*−1*	−1
16	−2.247578	−1	−1

Example: Iterative Decoding (cont.)

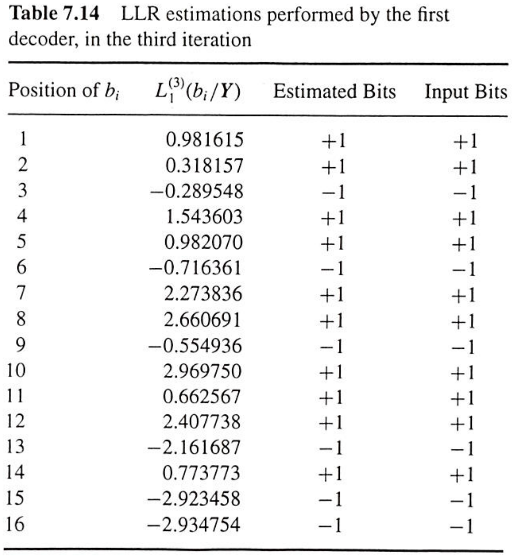


Table 7.14 LLR estimations performed by the first decoder, in the third iteration

Position of b_i	$L_1^{(3)}(b_i/Y)$	Estimated Bits	Input Bits
1	0.981615	+1	+1
2	0.318157	+1	+1
3	−0.289548	−1	−1
4	1.543603	+1	+1
5	0.982070	+1	+1
6	−0.716361	−1	−1
7	2.273836	+1	+1
8	2.660691	+1	+1
9	−0.554936	−1	−1
10	2.969750	+1	+1
11	0.662567	+1	+1
12	2.407738	+1	+1
13	−2.161687	−1	−1
14	0.773773	+1	+1
15	−2.923458	−1	−1
16	−2.934754	−1	−1

Interleavers for Turbo Codes

Interleavers, which permute positions in a stream of symbols, are useful for several purposes in coding, for example, to

- combat burst errors by distributing adjacent symbols (and thereby errors) among many words;
- obtain statistically independent sequences of symbols for turbo encoders.

Major types are *block*, *convolutional*, *random*, and *linear* interleavers.

Interleavers for Turbo Codes

Interleavers, which permute positions in a stream of symbols, are useful for several purposes in coding, for example, to

- combat burst errors by distributing adjacent symbols (and thereby errors) among many words;
- obtain statistically independent sequences of symbols for turbo encoders.

Major types are *block*, *convolutional*, *random*, and *linear* interleavers.

Interleavers for Turbo Codes

Interleavers, which permute positions in a stream of symbols, are useful for several purposes in coding, for example, to

- combat burst errors by distributing adjacent symbols (and thereby errors) among many words;
- obtain statistically independent sequences of symbols for turbo encoders.

Major types are *block*, *convolutional*, *random*, and *linear* interleavers.

Interleavers for Turbo Codes

Interleavers, which permute positions in a stream of symbols, are useful for several purposes in coding, for example, to

- combat burst errors by distributing adjacent symbols (and thereby errors) among many words;
- obtain statistically independent sequences of symbols for turbo encoders.

Major types are *block*, *convolutional*, *random*, and *linear* interleavers.

Block Interleavers

In a block interleaver, the symbols are written row-wise into an $M \times N$ matrix and read column-wise. With a table

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

the symbols are output in the order $1 \to 6 \to 11 \to 16 \to 2 \to \cdots$.

Design rule: Let $M = N$ and let M and N be odd.

Convolutional Interleavers

A convolutional interleaver is implemented using N shift registers, of length $0, L, 2L, \ldots, (N-1)L$:

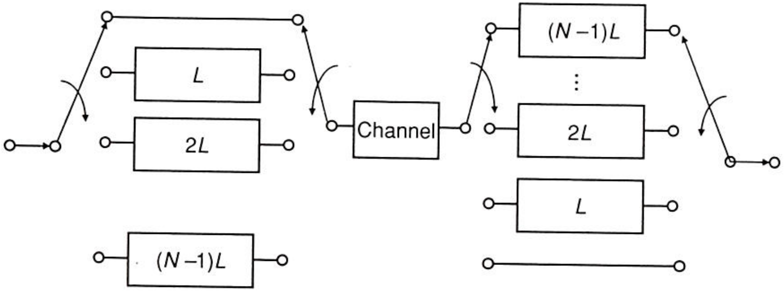


Figure 7.16 A convolutional interleaver

Convolutional Interleavers (cont.)

With $N = 3$ and $L = 1$, the symbols are output in the order $0 \to -2 \to -4 \to 3 \to 1 \to -1 \to 6 \to 4 \to \cdots$.

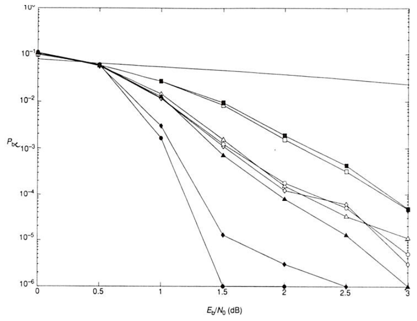
Easy implementation is a strength of convolutional interleavers. Note that no block of consecutive symbols is mapped to consecutive symbols.

Random Interleavers

A random interleaver gives a random permutation of a block of symbols. The permutation of a (pseudo-)random interleaver needs to be stored in memory.

- Performance improves when interleaver size increases.
  Note. Applications do not always allow large interleavers, implying long delays.
- If the interleaver size is small, then block interleavers are generally better than random interleavers.
- If the interleaver size is large, then random interleavers are generally better than block interleavers.

Example: Imapct of Interleaver



□——□	Block Int. 13x13, 8952 transmitted blocks
■——■	Random Int. 13x13=169, 8952 transmitted blocks
△——△	Block Int. 31x31, 1575 transmitted blocks
▲——▲	Random Int. 31x31=961, 1575 transmitted blocks
◇——◇	Block Int. 81x81, 230 transmitted blocks
◆——◆	Random Int. 81x81=6561, 230 transmitted blocks
○——○	Block Int. 123x123, 100 transmitted blocks
●——●	Random Int. 123x123=15,129, 100 transmitted blocks

Figure 7.17 BER performance of a turbo code as a function of the type and size of the interleaver

Linear Interleavers

A permutation of an interleaver given by a concise mathematical formula can save a lot of memory. A linear interleaver of length L maps a symbol in position i, $0 \leq i \leq L-1$, into position

$$pi + s \pmod{L}.$$

for some $0 \leq p, s \leq L-1$. It is required that $\gcd(p, L) = 1$.

A Practical Aspect of Decoding

Implementing decoding algorithms (for turbo and other codes) often means handling values of very different orders of magnitude $\Rightarrow$ overflow and underflow problems may occur.

Solution: Convert the values as well as the calculations into logarithmic form. Then

$$\begin{aligned} e^A \times e^B &= e^{A+B}, \\ e^A + e^B &= e^{\max(A,B)+\ln(1+e^{-|A-B|})}. \end{aligned}$$

Performance of Turbo Codes

The performance of turbo codes depends on the signal-to-noise ratio (SNR):

- At low values of SNR, iterative decoding performs worse than uncoded transmission, even for a large number of iterations.
- At low to medium values of SNR, iterative decoding is very effective. The performance increases with an increase in the number of iterations.
- A medium to high values of SNR, iterative decoding converges in few iterations. Performance increases only slowly as SNR increases.

Performance of Turbo Codes

The performance of turbo codes depends on the signal-to-noise ratio (SNR):

- At low values of SNR, iterative decoding performs worse than uncoded transmission, even for a large number of iterations.
- At low to medium values of SNR, iterative decoding is very effective. The performance increases with an increase in the number of iterations.
- A medium to high values of SNR, iterative decoding converges in few iterations. Performance increases only slowly as SNR increases.

Performance of Turbo Codes

The performance of turbo codes depends on the signal-to-noise ratio (SNR):

- At low values of SNR, iterative decoding performs worse than uncoded transmission, even for a large number of iterations.
- At low to medium values of SNR, iterative decoding is very effective. The performance increases with an increase in the number of iterations.
- A medium to high values of SNR, iterative decoding converges in few iterations. Performance increases only slowly as SNR increases.

Performance of Turbo Codes

The performance of turbo codes depends on the signal-to-noise ratio (SNR):

- At low values of SNR, iterative decoding performs worse than uncoded transmission, even for a large number of iterations.
- At low to medium values of SNR, iterative decoding is very effective. The performance increases with an increase in the number of iterations.
- A medium to high values of SNR, iterative decoding converges in few iterations. Performance increases only slowly as SNR increases.

Background

In addition to turbo codes, LDPC codes is a class of codes decoded iteratively and with good practical performance. LDPC codes were originally discovered by Gallager in the early 1960s and rediscovered by MacKay and Neal in 1996.

LDPC codes are occasionally called **Gallager codes**.

LDPC Codes

Low-density parity check (LDPC) codes are

- linear block codes with
- a sparse parity check matrix H.

Sparse means that most of the elements are 0. Note that the direction of constructing matrices is opposite to the normal one: design H and then calculate a generator matrix G, not design G and then calculate H.

LDPC Codes

Low-density parity check (LDPC) codes are

- linear block codes with
- a sparse parity check matrix H.

Sparse means that most of the elements are 0. Note that the direction of constructing matrices is opposite to the normal one: design H and then calculate a generator matrix G, not design G and then calculate H.

LDPC Codes

Low-density parity check (LDPC) codes are

- linear block codes with
- a sparse parity check matrix H.

Sparse means that most of the elements are 0. Note that the direction of constructing matrices is opposite to the normal one: design H and then calculate a generator matrix G, not design G and then calculate H.

LDPC Codes

Low-density parity check (LDPC) codes are

- linear block codes with
- a sparse parity check matrix H.

Sparse means that most of the elements are 0. Note that the direction of constructing matrices is opposite to the normal one: design H and then calculate a generator matrix G, not design G and then calculate H.

LDPC Codes (cont.)

regular LDPC code An LDPC code with constant number of 0s per row and per column.

irregular LDPC code An LDPC code that is not regular.

Alphabet for LDPC codes: GF(2), GF(4), GF(8), GF(16),. . . . Generally: better performance with bigger alphabet.

(Pseudo-)randomness occurs in turbo and LDPC codes in the interleaver and in the parity check matrix, respectively.

Irregular LDPC codes in general perform better than regular LDPC codes.

LDPC Codes (cont.)

regular LDPC code	An LDPC code with constant number of 0s per row and per column.
irregular LDPC code	An LDPC code that is not regular.

Alphabet for LDPC codes: GF(2), GF(4), GF(8), GF(16),. . . . Generally: better performance with bigger alphabet.

(Pseudo-)randomness occurs in turbo and LDPC codes in the interleaver and in the parity check matrix, respectively.

Irregular LDPC codes in general perform better than regular LDPC codes.

Tanner Graph

bipartite graph A graph with vertex set $V = V_1 \cup V_2$, where each edge has one endpoint in V_1 and one in V_2.

A **Tanner graph** of an LDPC code with parity check matrix H has one vertex in V_1 for each row of H and one vertex in V_2 for each column of H, and there is an edge between two vertices i and j exactly when $h_{ij} \neq 0$.

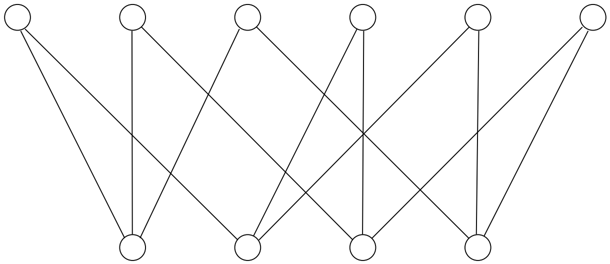
Tanner Graph

bipartite graph A graph with vertex set $V = V_1 \cup V_2$, where each edge has one endpoint in V_1 and one in V_2.

A **Tanner graph** of an LDPC code with parity check matrix H has one vertex in V_1 for each row of H and one vertex in V_2 for each column of H, and there is an edge between two vertices i and j exactly when $h_{ij} \neq 0$.

Example: Tanner Graph

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



Cycles of Tanner Graphs

Short cycles of Tanner graphs have a negative impact on decoding. Cycles necessarily have even length and length 2 is not possible.

Avoiding cycles of length 4: The intersection of positions in which two columns have nonzero values should be at most 1.

The requirement that a Tanner graph should not have short cycles is an intricate part in the construction of good LDPC codes.

Note. The degrading effect of short-length cycles diminishes as the code length increases and is strongly reduced with length > 1000 bits.

Obtaining Generator Matrices

To obtain the generator matrix, the parity check matrix is converted into systematic form—for example, using Gaussian elimination—after which the transformation is straightforward.

LDPC Code Types

LDPC codes can be divided into

- random LDPC codes and
- structured LDPC codes.

The best known codes are of the former types. Structured LDPC codes can be constructed from various types of combinatorial objects (designs, geometries,. . .).

LDPC Code Types

LDPC codes can be divided into

- random LDPC codes and
- structured LDPC codes.

The best known codes are of the former types. Structured LDPC codes can be constructed from various types of combinatorial objects (designs, geometries,. . .).

Decoding LDPC Codes

Decoding LDPC codes is an iterative process of interchanging information between the two types of nodes of the corresponding Tanner graph. If

- at some point of the iterative process the syndrome of the estimated decoded vector is the all-zero vector, this result is output;
- the iterative process has not converged to a solution after a predetermined number of iterations, decoding failure is declared.

See [MF, Fig. 8.5] for the impact of the maximum number on iterations on the BER performance.

Example: Impact of Number of Iterations

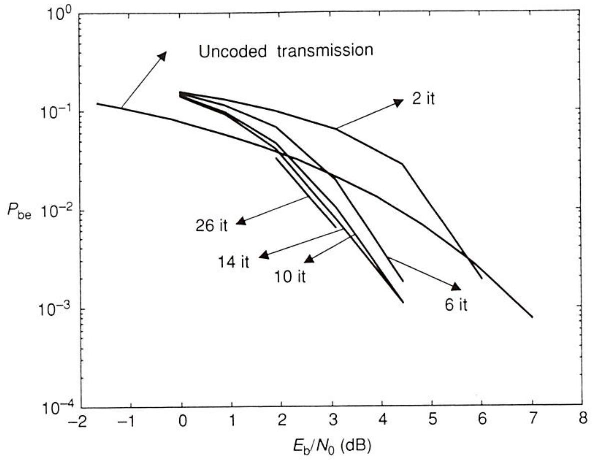


Figure 8.5 BER performance of the irregular LDPC code $C_b(60, 30)$ of rate $R_c = 1/2$, as a function of the number of iterations

Calculating Estimates

The core part of the turbo decoding algorithm in the previous lecture is the BCJR algorithm. The core part of the LDPC decoding algorithm is the **sum-product algorithm**, or **belief propagation algorithm**.

These algorithms are maximum a posteriori (MAP) algorithm—recall that the version of the Viterbi algorithm considered earlier is a maximum likelihood (ML) algorithm.

Some Practical Aspects of Decoding

- Polar format should be used instead of binary format.
- With logarithmic calculation, products and divisions are converted into additions and subtractions, respectively (cf. turbo coding slides).
- Look-up tables for parts of the logarithmic calculations save a lot of time and do not have a significant impact on the BER performance.

Some Practical Aspects of Decoding

- Polar format should be used instead of binary format.
- With logarithmic calculation, products and divisions are converted into additions and subtractions, respectively (cf. turbo coding slides).
- Look-up tables for parts of the logarithmic calculations save a lot of time and do not have a significant impact on the BER performance.

Some Practical Aspects of Decoding

- Polar format should be used instead of binary format.
- With logarithmic calculation, products and divisions are converted into additions and subtractions, respectively (cf. turbo coding slides).
- Look-up tables for parts of the logarithmic calculations save a lot of time and do not have a significant impact on the BER performance.

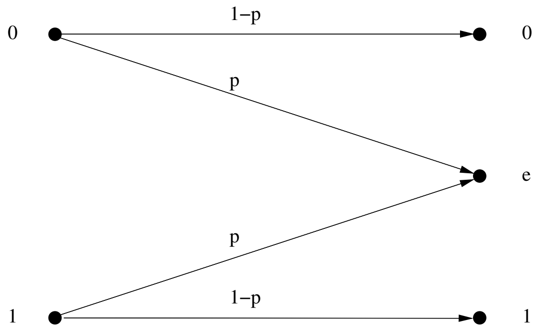
Turbo vs. LDPC Codes

Turbo codes: Very good BER performance for intermediate block length.

LDPC codes: Very good BER performance for long block length (for example, BER performance with length $n = 10,000$ that is less than 0.1 dB from the Shannon limit).

Erasure Channel

The code type to be discussed next is designed for the **erasure channel** rather than AWGN, BSC, and similar channels assumed so far.



Encoding Fountain Codes

Consider a (binary) message consisting of K data packets of m bits each, that is, of total length Km. (It is common that communication protocols transmit information in packets.)

Fountain code: The transmitter continuously transmits packets of m bits that are obtained by XORing—that is, adding modulo 2—subsets of the packets. The receiver collects (just a little bit more than) K packets to retrieve the original message.

Decoding Fountain Codes

Decoding fountain codes means solving a system of equations. Call the blocks $B_1, B_2, \ldots, B_K$.

Example. $K = 3$, $m = 4$. Assume that for the received blocks we have

$$\begin{aligned} B_1 + B_3 &= 0100, \\ B_2 + B_3 &= 1110, \\ B_1 + B_2 + B_3 &= 0000. \end{aligned}$$

Adding the second and the third equation gives $B_1 = 1110$, and then $B_3 = 1010$ and $B_2 = 0100$.

Random Fountain Codes

How to form the packets? One possibility: Random combinations/sums of packets. The indices of the packets involved must be known also by the receiver. When N such packets have been received

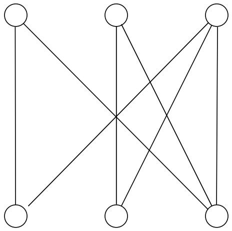
- if $N < K$, decoding is not possible,
- if $N = K$, decoding is possible with probability about 0.289,
- if $N = K + \Delta$, decoding is possible with probability at least $1 - 2^{-\Delta}$.

Bipartite Graphs for Fountain Codes

Decoding fountain codes is about solving a system of equations, which can be rather time-consuming if K is large.

As with LDPC codes, a bipartite graph may be useful in the decoding process, with one set of nodes corresponding to the blocks (variables) and the other to the received words.

Example. (cont.)



Luby Transform Codes

Luby transform (LT) codes are improved random fountain codes.

- Random combinations/sums have only a few packets.
- The number of packets in the sums are given by an optimized distribution function.
- Decoding is straightforward due to equations of the form $B_i = ?$ throughout the calculations.

Luby Transform Codes

Luby transform (LT) codes are improved random fountain codes.

- Random combinations/sums have only a few packets.
- The number of packets in the sums are given by an optimized distribution function.
- Decoding is straightforward due to equations of the form $B_i = ?$ throughout the calculations.

Luby Transform Codes

Luby transform (LT) codes are improved random fountain codes.

- Random combinations/sums have only a few packets.
- The number of packets in the sums are given by an optimized distribution function.
- Decoding is straightforward due to equations of the form $B_i =?$ throughout the calculations.

Luby Transform Codes

Luby transform (LT) codes are improved random fountain codes.

- Random combinations/sums have only a few packets.
- The number of packets in the sums are given by an optimized distribution function.
- Decoding is straightforward due to equations of the form $B_i =?$ throughout the calculations.

Background

In the basic coding framework, one assumes that there is one sender and one receiver, and if there are intermediate nodes in the network between the sender and the receiver, these simply forward the packets that they receive.

However, with more than one sender and/or more than one receiver, and a network of intermediate nodes processing packets actively, information can often be transmitted at a higher rate than in the basic store-and-forward setting. This topic has been coined **network coding**.

References

The term **network coding** was coined in

R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, Network information flow, *IEEE Transactions on Information Theory* **46** (2000), 1204–1216.

Introductory text to this topic can be found in

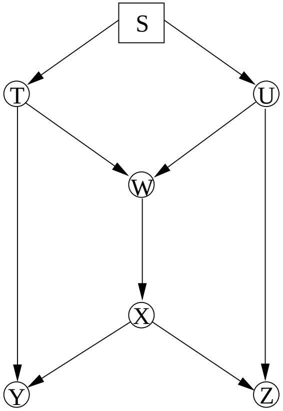
[YLCZ] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, Network Coding Theory,

available electronically at

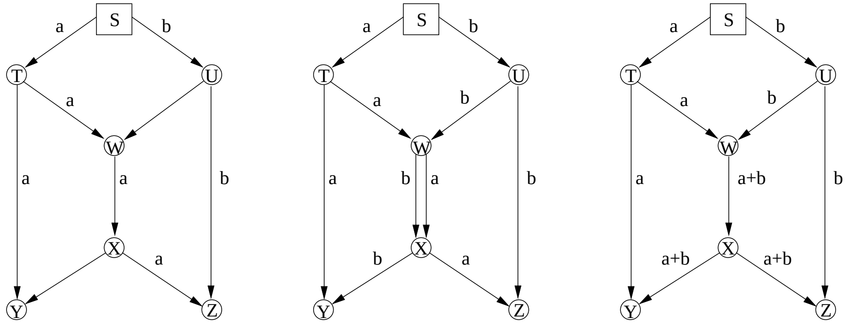
`http://iest2.ie.cuhk.edu.hk/~whyeung/publications/tutorial.pdf`

The Butterfly Network

The standard example network to demonstrate network coding is called the **butterfly network**.



Multicasting over a Communication Network



Properties of the Various Solutions

- The first method transmits 1.5 bits per receiver and transmission.
- The second and the third method transmit 2 bits per receiver and transmission.
- The second method uses 10 channels within the network, whereas the third method uses 9 channels.

Network coding minimizes energy consumption and maximizes bit rate.

Example: Satellite Communication

Two ground stations can communicate with each other through a satellite in the following way:

- The ground stations send the packets *a* and *b*, respectively to the satellite.
- The satellite sends $a + b$ back.
- The ground stations decode the unknown part.
- Simple coding, yet the downlink bandwidth can be reduced by 50%.

Example: Satellite Communication

Two ground stations can communicate with each other through a satellite in the following way:

- The ground stations send the packets *a* and *b*, respectively to the satellite.
- The satellite sends $a + b$ back.
- The ground stations decode the unknown part.
- Simple coding, yet the downlink bandwidth can be reduced by 50%.

Example: Satellite Communication

Two ground stations can communicate with each other through a satellite in the following way:

- The ground stations send the packets a and b, respectively to the satellite.
- The satellite sends $a + b$ back.
- The ground stations decode the unknown part.
- Simple coding, yet the downlink bandwidth can be reduced by 50%.

Example: Satellite Communication

Two ground stations can communicate with each other through a satellite in the following way:

- The ground stations send the packets a and b, respectively to the satellite.
- The satellite sends $a + b$ back.
- The ground stations decode the unknown part.
- Simple coding, yet the downlink bandwidth can be reduced by 50%.

Example: Satellite Communication

Two ground stations can communicate with each other through a satellite in the following way:

- The ground stations send the packets a and b, respectively to the satellite.
- The satellite sends $a + b$ back.
- The ground stations decode the unknown part.
- Simple coding, yet the downlink bandwidth can be reduced by 50%.

Example: Wireless Relay Stations

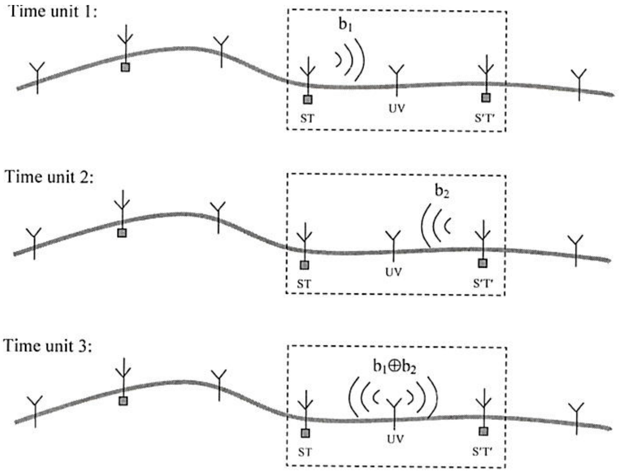


Fig. 1.3 Operation of the relay transceiver between two wireless base stations.

Error Control Systems with Feedback

So far, only *forward error correction* (FEC) has been considered. With information flow in both directions of a channel, other options become available. The most important error control methods for channels with feedback are as follows:

1. Automatic-repeat-request (ARQ) protocols.
2. Type-I hybrid-ARQ protocols.
3. Type-II hybrid-ARQ protocols.

Pure ARQ Protocols

The objective of a pure ARQ protocol is a system that will detect an error burst, discard the affected packet (the message is broken up into *packets* of length k), and request a retransmission.

The most frequently used error-detecting codes in ARQ protocols are the CRC codes.

Performance Measures for ARQ Protocols

The following two measures are often used to evaluate the performance of an ARQ protocol.

accepted packet error rate The percentage of packets accepted by the receiver that contain one or more bit/symbol errors.

throughput (η) The average number of encoded *n*-bit data packets accepted by the receiver in the time it takes the transmitter to send a single unencoded *k*-bit data packet.

Average number of transmissions

Let P_r be the probability that a retransmission request is generated for a received packet. If the random variable T is defined as the number of times a packet must be transmitted before it is accepted, then according to Eq. 15-3 in [Wic] its expectation T_r is given by

$$T_r = \mathrm{E}[T] = \frac{1}{1 - P_r}.$$

This expression is useful in calculating the throughputs of the basic ARQ protocols.

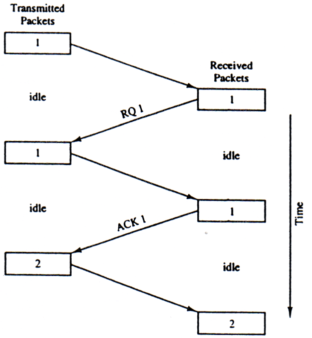
Retransmission Protocols

In selecting a retransmission protocol, the designer must strike a balance between the complexity of the design and the throughput of the resulting system. The three basic retransmission protocols are the following:

1. Stop-and-wait (SW-ARQ).
2. Go-back-N (GBN-ARQ).
3. Selective repeat (SR-ARQ).

Stop-and-Wait (1)

In the stop-and-wait (SW-ARQ) protocol, the transmitter sends out a packet and waits for an acknowledgment. The receiver responds by sending an acknowledgment (ACK) if the packet was deemed error-free, or it sends a retransmission request (RQ) if the packet contained a detectable error pattern; see [Wic, Fig. 15-1]. The transmitter is idle while waiting for the acknowledgment.



Stop-and-Wait (2)

The primary benefit of the SW-ARQ protocol is that there is no need for packet buffering at the transmitter or receiver. The primary disadvantage of the protocol is that it provides poor throughput, in particular when the propagation delays are long (satellite communications, etc.).

If Γ denotes the number of bits that could be transmitted during the idle time of the transmitter, then the throughput of the SW-ARQ protocol is

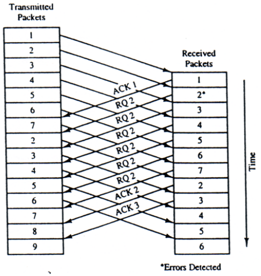
$$\eta_{SW} = \frac{k}{T_r(n+\Gamma)} = R\left(\frac{1-P_r}{1+\Gamma/n}\right),$$

where $R = k/n$ is the rate of the error-detecting code used in the protocol and Γ is the number of bits that could have been transmitted during the idle time.

Go-Back-N (1)

If we are willing to allow for some buffering in the transmitter, the go-back-N (GBN-ARQ) protocol can be implemented; see [Wic, Fig. 15-2]. In this protocol the transmitter sends packets in a continuous stream.

When the receiver detects an error, it sends an RQ for that packet and waits for its second copy (and ignores all subsequent packets until the second packet is received, so receiver buffering is avoided). The transmitter responds by resending the requested packet and *all subsequent packets* (so buffering is necessary in the sender).



Go-Back-N (2)

Recall that Γ denotes the number of bits that can be transmitted during the idle time. For the required size of the buffer, N, we get that

$$N = \left\lceil \frac{\Gamma}{n} \right\rceil + 1,$$

since the the number of packets that can be transmitted (at least partially) during the idle time is $\lceil \Gamma / n \rceil$ (note that we disagree slightly with Eq. 15-6 in [Wic]). The throughput of the GBN-ARQ protocol is

$$\eta_{GBN} = \left(\frac{k}{n}\right)\left(\frac{1}{1+(T_r-1)N}\right) = R\left(\frac{1-P_r}{1+P_r(N-1)}\right).$$

Selective Repeat

If we allow for buffering in both the transmitter and the receiver, we can implement a selective-repeat (SR-ARQ) protocol; see [Wic, Fig. 15-3]. The transmitter sends a continuous stream of packets and responds to retransmission requests by sending the requested packet and thereafter resuming the transmission where it was stopped.

The throughput of the SR-ARQ protocol is simply

$$\eta_{SR} = \left(\frac{k}{n}\right)\left(\frac{1}{T_r}\right) = R(1 - P_r).$$

Noisy Feedback Channels

The following additional protocols can be implemented for a noisy feedback channel.

- ▷ Each time the transmitter sends out a packet, a timer for that packet is started. If a response is not obtained for that packet after a given period of time, it is assumed that the response is an RQ.
- ▷ When the receiver sends an RQ, the receiver starts a timer for that RQ. If a new copy of the packet is not received after a given period of time, the RQ is sent again.
- ▷ If the receiver receives a packet that has already been accepted, an ACK is sent and the packet is discarded.

Hybrid-ARQ Protocols

In hybrid-ARQ protocols, each packet is encoded for both error detection and error correction. The FEC portion corrects the most common error patterns. Hybrid protocols provide throughput similar to that of FEC systems, while offering reliability performance typical of ARQ protocols.

Hybrid-ARQ protocols are divided into the simpler type-I protocols and the more advanced type-II protocols.

Type-I Hybrid-ARQ Protocols (1)

Type-I hybrid-ARQ protocols can be implemented using either one-code or two-code systems. In a two-code system, the data is first encoded using a high-rate error-detecting code (CRC codes are frequently used). The encoded data is then encoded once again using a FEC code.

An example of a single-code type-I hybrid-ARQ protocol is given in [Wic, Example 15-4]. In [Wic, Fig. 15-11] and [Wic, Fig. 15-12] the performance of a pure ARQ protocol and a type-I hybrid-ARQ protocol are compared with respect to reliability and throughput.

Type-I Hybrid-ARQ Protocols (2)

The single-code type-I hybrid-ARQ protocols are based on the following idea.

If the minimum distance of a linear block code satisfies the condition

$$d_{\min} \geq \lambda + l + 1,$$

then the code is capable of correcting all error patterns with λ or fewer errors and simultaneously detecting all error patterns with l ($l > \lambda$) or fewer errors.

Packet Combining Systems

These systems offer adaptive code rates for different channel conditions. Two distinct types:

code combining The individual transmissions (of one data block) are encoded at some rate R. If the receiver has N packets that have caused retransmission requests, these packets are concatenated to form a single packet encoded at rate R/N.

diversity combining Multiple identical copies of a packet are used to create a single packet whose symbols are more reliable than those of any of the individual copies. Symbol voting is used in hard-decision systems and symbol averaging in soft-decision systems.

Type-II Hybrid-ARQ Protocols (1)

Type-II hybrid-ARQ protocols are code combining systems where N (the number of combined packets) is limited to 2. They adapt to changing channel conditions through the use of *incremental redundancy*. The transmitter in these systems responds to retransmission requests by sending additional parity bits to the receiver. These additional bits allow for increased error-correction capability.

Type-II Hybrid-ARQ Protocols (2)

The “original” type-II hybrid-ARQ protocol proposed by Wang and Lin:

1. A k-bit message is first encoded using a high-rate (n, k) error detecting code C_1 to form an n-bit packet P_1.

2. P_1 is encoded using a $(2n, n)$ systematic *invertible* code C_2.

3. The n parity bits (denoted by P_2) from the C_2 codeword are saved in a buffer, while the C_1 codeword P_1 is transmitted.

4. If the initial transmission of P_1 is found to contain errors, it is stored in a buffer, a retransmission request is sent, and the transmitter responds by sending P_2.

A code is **invertible** if the parity-check symbols can be used by themselves to uniquely determine the information symbols.

Type-II Hybrid-ARQ Protocols (3)

5. The received version of P_2 is first inverted and checked for errors. If there are detected errors, the received noisy version of P_2 is appended to the earlier received (also noisy) version of P_1 to form a noise-corrupted C_2 codeword. After decoding, the resulting *n*-bit word is again checked for errors using C_1. If errors are detected, the process continues, with the transmitter alternating transmission of P_1 and P_2 until the decoding/ error detection process is successful.

For example, shortened half-rate cyclic codes have been used as C_2-codes.