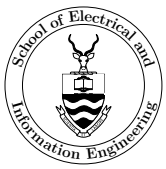


A Short Course on Error Control Coding (Telecommunications Access Network)

by

Ling Cheng

School of Electrical and Information Engineering

at the

University of the Witwatersrand

October, 2010

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

Nowadays, digital communication is an integral part of our lives. Error correcting codes play an important role in the design of a successful digital communication system. Since Shannon [1] proved the channel coding theorem in 1948, often called the Shannon limit, it has prompted more studies on the design of stronger error correcting codes with more complexity. More and more close-to-bound error correcting codes, e.g. turbo codes and low-density parity-check codes, were discovered or rediscovered in the past two decades.

In this course, we will present a number of important error control coding (ECC) techniques which are widely used in telecommunications system. We attempt to maintain the balance between the mathematics and their practical implications on telecommunications.

In Section 1.2, we will first illustrate the problems that will be investigated in this work, and the goals that this work aims for. Then, in Section 1.3, we will illustrate the framework of a communication system. The Shannon limit will be introduced in Section 1.4. In Section 1.5, we will present the layout of this article.

1.2 Problem Statement

In most communication systems, errors are introduced through the data exchange, processing and storage (*channel*). How to reproduce highly reliable data from a unreliable channel becomes a major concern for the system designer. The main problem investi-

gated in this article is how to find good error correcting codes to protect the system from *substitution* errors. More specifically, how can we effectively retrieve original sequences sent when there are random substitution errors, and how can we increase the transmission rate with the least trade-off of the system complexity at the same time?

To answer these questions, we will first introduce channel capacity and the landmark Shannon limit [1].[1] In this course, we will present two basic coding techniques, block codes and convolutional codes. We will also present the basics of some state-of-the-art coding techniques, i.e., low-density parity-check and soft decision. As known, based on the techniques presented in this course, channel coding can already offer performance close to theoretical bounds.

1.3 Framework of Communications System

In Fig. 1.1, we present the system model of a communication system, which is the classic mathematical model of communication introduced by Shannon and Weaver in [2].

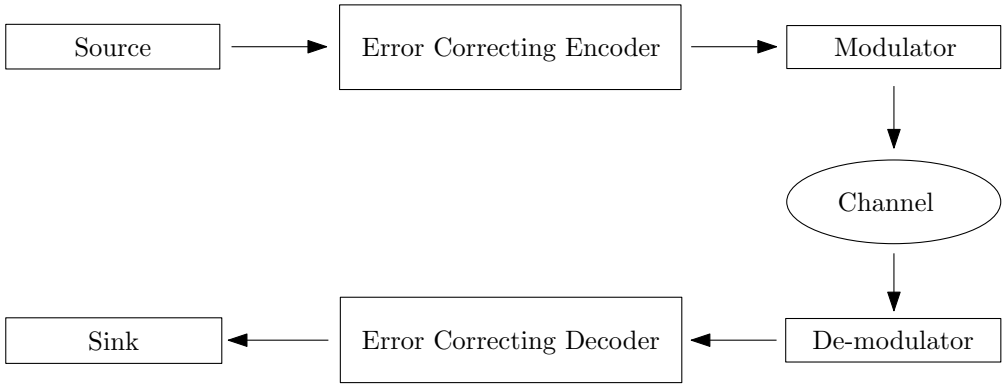


Figure 1.1: System model of a communication system.

Regarding the system model, we have some important issues to clarify:

First, unless stated otherwise, we only consider a forward error correction (FEC) system. In practice, a communication system can be FEC and/or automatic-repeat-request (ARQ).

[1]Channel capacity is defined as the maximal code rate under a certain channel state, which can guarantee a reliable data transmission.

Second, we want to clarify that a practical communication system includes more steps and involves more coding techniques, *e.g.*, source coding and modulation coding. The introduction to these techniques is not included in this article, and can be obtained from relevant textbooks.

In this classic model, the information transmitted from the source to its destination is an abstract conception. In the real world, it can be images, computer data, voice, and so forth.

To achieve minimized probability of erroneous transmission, an error protection scheme might be used. This protection can be achieved through either of the following strategies: forward error correction (FEC) or automatic repeat request (ARQ). FEC can be implemented by using error correction coding.

1.4 The Shannon Limit and Channel Coding

In 1948, Shannon [1] proved that for a band-limited additive white Gaussian noise channel with bandwidth B, there exists families of coding schemes that can achieve an arbitrarily small probability of error at the receiving end at a communication rate less than the capacity of the channel, which can be described as follows:

$$C = B\log_2(1+\frac{S}{N})\text{bits/sec}, \tag{1.1}$$

where C represents the capacity of the channel, S and N are the average signal power and noise power. The implication of (1.1) is that, if the information rate can be dropped below the capacity of the transmission channel, with proper error protection means, such as error correcting codes, error free transmission is possible.

The data transmission or storage can be protected from the errors induced by a noise channel or storage medium, provided that the information transmit rate R is less than the capacity C,

$$R < C. \tag{1.2}$$

Example 1.1 *Since*

$$N = N_0 B, \tag{1.3}$$

if the two-sided power spectrum density of the noise is $N_0/2$ watts/Hz, we can obtain

$$C = B\log_2(1+\frac{S}{N_0B}) \tag{1.4}$$

$$= B\log_2(1+\frac{CE_b}{N_0B}), \tag{1.5}$$

where E_b denotes the energy per bit.

The definition of the spectral efficiency η is as follows:

$$\eta = \frac{C}{B}. \tag{1.6}$$

Therefore, from (1.5) *and* (1.6)

$$\eta = \log_2(1+\eta\frac{E_b}{N_0}). \tag{1.7}$$

For BPSK modulation, the spectral efficiency is $\eta = 1$. The corresponding E_b/N_0 is 1 (0 dB). It means on the basis of a coding scheme, a BPSK-modulation system can achieve error-free transmission over an AWGN channel at signal-noise-ratio per bit 0 dB.

We assume a unlimited bandwidth, in other words, $B \to \infty$ and $\eta = 0$. From (1.7),

$$\frac{E_b}{N_0} = \frac{2^\eta - 1}{\eta}. \tag{1.8}$$

Therefore, we have

$$\lim_{\eta\to 0} E_b/N_0 = \ln 2, \tag{1.9}$$

which is -1.6 dB.

Example 1.2 *Not considering its bandwidth efficiency, binary phase shift keying (BPSK) is known to be the optimum binary modulation scheme, because it is one kind of antipodal sampling. Consequently, it is often used in communications theory as a benchmark for comparison. For uncoded BPSK, we have*

$$BER = Q(\sqrt{2E_b/N_0}), \tag{1.10}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-y^2/2} dy, \tag{1.11}$$

and BER denotes the bit error rate of the received sequence. The result is illustrated by Fig. 1.2.

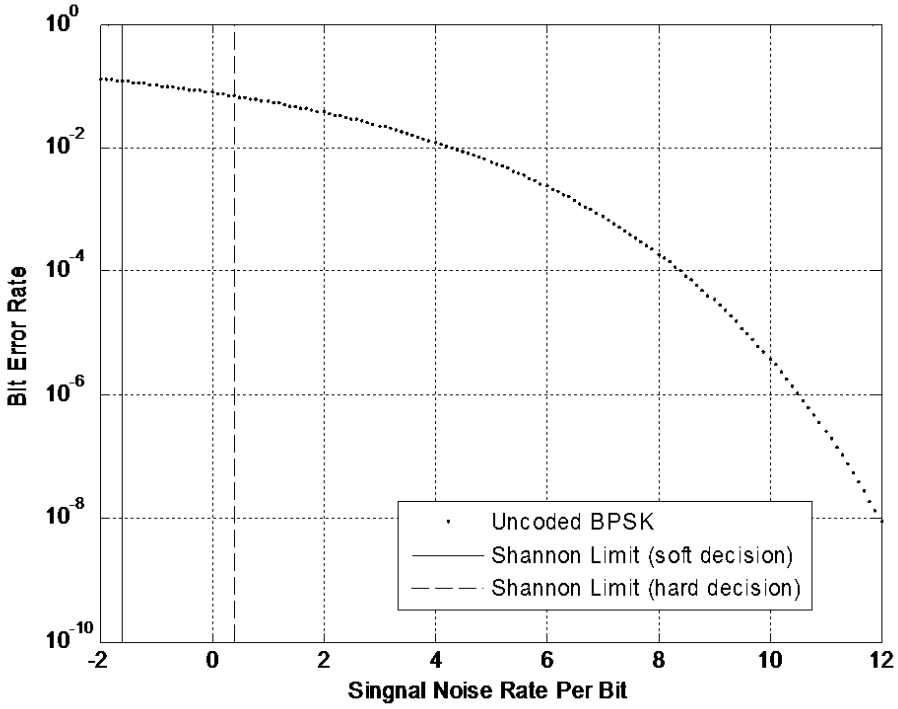


Figure 1.2: Shannon Limit and Performance of an Uncoded BPSK system

These breakthroughs of information theory prompted academics to look for explicit error control codes. A lot of research was expended on designing efficient and reliable encoders and decoders. This research introduced error control coding as an important aspect of the design of the digital communication or storage system. In 1949, Golay and in 1950 Hamming introduced different kinds of practical error control codes, which all belong to block codes. A decade later, a class of longer block codes, known as BCH codes, was found by Bose, Ray-Chaudhuri and Hocquenghem. Convolutional codes were first introduced by Peter Elias in 1955. In 1960, Reed and Solomon designed the Reed-Solomon codes, which can provide distinguished error correction performance and are introduced specifically to protect against burst errors existing in transmission. The performance of a most popular concatenated coding system using a Viterbi-decoded convolutional inner

code and a Reed-Solomon outer code, with interleaving between these two coding steps, is only 4dB from the Shannon limit. After fifty years of research, in 1993 Berrou and Glavieux presented a coding scheme named turbo codes, which can provide a coding gain near to the Shannon limit. In 1962, Gallager introduced a class of linear block codes, named low-density parity-check (LDPC) codes [3]. In 1981, Tanner generalized LDPC codes and introduced Tanner graphs to represent LDPC in [4], based on which the belief propagation (BP) algorithm is found to be a good candidate decoding algorithm. In 1995, MacKay re-invented LDPC codes [5] and pointed out that LDPC codes can provide close-to Shannon limit performances.

1.5 Outline

A brief introduction and problem statement, as well as the outline of the rest of this article are provided in the current chapter.

Block codes, which is one of two major categories of error correcting codes will be introduced in Chapter 2. We will present Hamming codes, Reed-Solomon codes and low-density parity-check codes. In the same chapter decoding part, we will introduce syndrome decoding algorithm, Berlekamp-Massey decoding algorithm, Tanner graph and belief-propagation decoding algorithm.

The other major category of error correcting codes, convolutional codes will be introduced in Chapter 3. Following the code structures and properties, we will present the Viterbi decoding algorithm.

In Chapter 4, we will introduce interleaving and concatenated system.

2 Block Codes

Two years after Shannon published his landmark paper, Hamming in 1950 found the first class of linear block codes for error correction. In this chapter, we will first present the basics of linear block codes in Section 2.1. Then we will introduce several classes of important linear block codes and a few decoding algorithms. In Section 2.2, we first introduce Hamming codes. Then, a very brief introduction to the linear algebra is presented in Section 2.3. In Section 2.4, we will continue to present Reed-Solomon codes; and in Section 2.5, the Berlekamp-Massey decoding algorithm is introduced. Low-density parity-check codes will be presented in Section 2.6; Tanner graph and belief-propagation decoding algorithm will be introduced in Section 2.7. In Section 2.8 we will present how to construct a very long LDPC code, which can provide close-to-Shannon-limit performance.

2.1 Basics of Linear Block Codes

A linear (n, k) block code always can be presented as

$$\mathbf{v} = \mathbf{uG}. \tag{2.1}$$

For binary codes, $\mathbf{u}$ can be any bit sequences of length k, and $\mathbf{v}$ is the unique corresponding codeword of length n. Here G has k linear independent rows and n columns, and is called the *generator matrix*. If

$$\mathbf{G} := \begin{pmatrix} \mathbf{P} & \mathbf{I}_k \end{pmatrix} \tag{2.2}$$

the linear block code is systematic. Here $\mathbf{I}$ is a $k \times k$ identity matrix. By a systematic encoding, the information $\mathbf{u}$ is part of the codeword $\mathbf{v}$.

So what is the *parity-check matrix*? Given a generator matrix $\mathbf{G}$, there exists an $(n-k) \times n$ matrix $\mathbf{H}$ *iff.* $\mathbf{vH}^T = \mathbf{0}$.

If $\mathbf{G}$ has the form as described by (2.2), $\mathbf{H}$ takes the following form:

$$\mathbf{H} := \begin{pmatrix} \mathbf{I}_{n-k} & \mathbf{P}^T \end{pmatrix} \tag{2.3}$$

Definition 2.1 *Hamming distance of two equal length sequences is the number of positions at which the corresponding symbols are different.*

Definition 2.2 *Minimum Hamming distance $d_{\min}$ of a code is the minimum value of the Hamming distances of any two codewords.*

2.2 Hamming Code

For any positive integer $m \geq 3$, there exists a Hamming code with

1) codeword length: $n = 2^m - 1$
2) length of information: $k = 2^m - m - 1$
3) length of parity bits: $n - k = m$
4) $d_{\min} = 3$

How to get the generator matrix of Hamming codes?

1) choose m;
2) enumerate all binary sequences of length m from $0 \ldots 01$ to $1 \ldots 1$;
3) create the parity-check matrix by filling these binary sequences in the matrix column by column in the form as described by (2.3);
4) obtain the generator matrix.

Example 2.1 *For $m = 3$, we first enumerate non-all-zero bits of length 3 such as*

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{2.4}$$

Then we can put the sequences column-wisely to create a parity-check matrix:

$$\mathbf{H} = \left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right) \tag{2.5}$$

Therefore, we can generate the generator matrix:

$$\mathbf{G} = \left(\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \tag{2.6}$$

Hamming code is a single-error-correcting code, since $d_{\min} = 3$. We assume one bit in the codeword $\mathbf{u}$ is changed through the transmission. Let $\mathbf{v}'$ denote the received sequence, and let $\mathbf{e}$ denote the single error. Note that $\mathbf{e}$ is a bit sequence of length n with a single one at the position that the error occurs. It is evident that

$$\mathbf{uG} = \mathbf{v} \tag{2.7}$$

$$\mathbf{v} + \mathbf{e} = \mathbf{v}'. \tag{2.8}$$

If we compare the Hamming distance of the received sequence $\mathbf{u}'$ with all codewords, we will find one codeword with $\mathbf{u}'$ has a Hamming distance 1; while all other codewords with $\mathbf{u}'$ have the Hamming distance no less than two. We can find and correct the codeword by using this method. However, it is not the easiest way to find the answer, because

the received sequence has to be compared to every codeword. Based on the following derivation, we will find an approach to reduce the decoding complexity:

$$(\mathbf{uG}+\mathbf{e})\mathbf{H}^T = \mathbf{vH}^T + \mathbf{eH}^T \tag{2.9}$$

$$= \mathbf{0} + \mathbf{eH}^T \tag{2.10}$$

$$= \mathbf{eH}^T \tag{2.11}$$

Since $\mathbf{e} = (0, \ldots, 0, 1, 0, \ldots, 0)$ and $\mathbf{H}^T$ can be presented as

$$\mathbf{H}^T = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_{2^m-1} \end{pmatrix}, \tag{2.12}$$

$$\mathbf{eH}^T = h_i, \tag{2.13}$$

where $1 \leq i \leq 2^m - 1$.

Actually, if we multiply the received sequence with the transport parity-check matrix, the result, called syndrome, will lead to the error location.

Example 2.2 *We use the same generator matrix as Example 2.1. Given information* $\mathbf{u} = 0001$*, we sent the codeword*

$$\mathbf{v} = \mathbf{uG} \tag{2.14}$$

$$= \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.16}$$

over the channel.

As the result of one error, we receive

$$\mathbf{v}' = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{2.17}$$

After the following calculation:

$$\mathbf{v}'\mathbf{H}^T = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{2.18}$$

$$= \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}, \tag{2.19}$$

we find the syndrome sequence is same as the fifth row of $\mathbf{H}^T$*. This leads to two discoveries:*

1) Detection: there is error in the received sequence

2) Correction: if there is only one error, it must appear at the fifth position.

Therefore, we change the received sequence into

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.20}$$

which is the correct codeword we send.

2.3 Irreducible, Primitive and Minimal Polynomials and Construction of the Galois Field

Definition 2.3 *If polynomial* f *can be described as* $f(x) = g(x)d(x)$*, and* $d(x)$ *is neither 1 nor* $f(x)$*,* f *is reducible over* K*; otherwise it is irreducible.*

Definition 2.4 *An irreducible polynomial of degree* r *is primitive, if it is not a divisor of* $1 + x^m$*, where* $m < 2^r - 1$*.*

Thus, according to Definition 2.4, using a primitive polynomial to construct mathematic computing in the $GF(2^r)$ field is convenient. And every non-zero world in $GF(2^r)$ can be represented as a power of a primitive element α.

Definition 2.5 *For any element α in $GF(2^r)$, the minimal polynomial is the polynomial with the smallest degree having α as a root.*

Example 2.3 *Construction of a $GF(2^4)$ field based on the primitive polynomial $h(x) = 1 + x + x^4$:*

TABLE 2.1: CONSTRUCTION OF A $GF(2^4)$ FIELD BY $h(x) = 1 + x + x^4$

Codeword	Polynomial in $x \pmod{h(x)}$	Power of α
0000	0	–
1000	1	1
0100	x	α
0010	x^2	α^2
0001	x^3	α^3
1100	$1 + x$	α^4
0110	$x + x^2$	α^5
0011	$x^2 + x^3$	α^6
1101	$1 + x + x^3$	α^7
1010	$1 + x^2$	α^8
0101	$x + x^3$	α^9
1110	$1 + x + x^2$	α^{10}
0111	$x + x^2 + x^3$	α^{11}
1111	$1 + x + x^2 + x^3$	α^{12}
1011	$1 + x^2 + x^3$	α^{13}
1001	$1 + x^3$	α^{14}

In this example, the minimal polynomial of each element is shown as follows:

TABLE 2.2: MINIMAL POLYNOMIALS OF THE ELEMENTS IN $GF(2^4)$

Elements of $GF(2^4)$ using $h(x)$	Minimal polynomial
0	x
1	$x + 1$
α, α^2, α^4, α^8	$x^4 + x + 1$
α^3, α^6, α^9, α^{12}	$x^4 + x^3 + x^2 + x + 1$
α^5, α^{10}	$x^2 + x + 1$
α^7, α^{11}, α^{13}, α^{14}	$x^4 + x^3 + 1$

2.4 Reed-Solomon Code

As one type of linear code, Reed-Solomon codes are a subset of non-binary BCH codes. Reed-Solomon codes work on the $GF(q)$ field, where q represents any power of one prime number p.

For any positive integer of s and t, there exists a q-ary BCH code of length $n = q^s - 1$, which is capable of correcting not more than t errors. The special subclass of q-ary t-error-correcting BCH codes for which $s = 1$ is called the Reed-Solomon codes, and the properties of Reed-Solomon codes are as follows:

1) block length: $n = q - 1$;

2) parity-check length: $n - k = 2t$;

3) $d_{\min} = 2t + 1$.

Consider Reed-Solomon codes with the alphabet from the Galois field $GF(2^r)$. The generator polynomial of a primitive t-error-correcting Reed-Solomon code of length $n = 2^r - 1$ is

$$g(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3) \dots (x + \alpha^{2t}) \tag{2.21}$$

Thus, we can obtain

$$g(x) = g_0 + g_1 x + g_2 x^2 + \dots + g_{2t-1} x^{2t-1} + x^{2t}. \tag{2.22}$$

Therefore, the generator polynomial has the roots of α, α^2,..., α^{2t}.

Let $g(x)$ have degree $n - k$. If $g(x)$ generates a linear cyclic code C over GF(2^r) of

length $n = 2^r - 1$, and dimension k then

$$G = \begin{pmatrix} g(x) \\ xg(x) \\ x^2 g(x) \\ \vdots \\ x^{k-1} g(x) \end{pmatrix} \tag{2.23}$$

is a generator matrix for C, and the number of codewords in C is $(2^r)^k$.

Thus, the codeword corresponding to any k-length symbol sequence $m(x)$ is mG.

2.5 Decoding of Reed-Solomon Codes and Berlekamp-Massey Algorithm

Since Reed-Solomon codes are over the $GF(q)$ field, the decoding process not only needs to look for the locations of errors, but also the magnitudes of errors. The error magnitude of an error location i is the element of $GF(q)$ that occurs in coordinate i of the (most likely) error pattern. For any binary code, since the 1 is the only non-zero element of $GF(2)$, the location of error can completely satisfy the determination of an error.

Thus, the first steps for decoding a Reed-Solomon code are the same as that of a binary BCH code. In addition, an extra step is required to determine the magnitude of error.

The Berlekamp-Massey algorithm has a fast performance on finding the error locator polynomial and is shown as follows:

Let w be a received word that was encoded using the RS(2^r, δ) code with generator

$$g(x) = (x + \alpha^{m+1})(x + \alpha^{m+2}) \dots (x + \alpha^{m+\delta-1}). \tag{2.24}$$

Let $t = \left\lfloor \frac{\delta-1}{2} \right\rfloor$.

1) Calculate $s_j = w(\beta^j)$ for $m + 1 \leq j \leq m + 2t$.

2) Define

$$q_{-1}(x) = 1 + s_{m+1}x + s_{m+2}x^2 + \ldots + s_{m+2t}x^{2t}, \tag{2.25}$$
$$q_0(x) = s_{m+1} + s_{m+2}x + \ldots + s_{m+2t}x^{2t-1}, \tag{2.26}$$
$$p_{-1}(x) = x^{2t+1}, \tag{2.27}$$
$$p_0(x) = x^{2t}. \tag{2.28}$$

Let $d_{-1} = -1$, $d_0 = 0$ and $z_0 = -1$.

3) For $1 \leq j \leq 2t$, define q_i, p_i, d_i and z_i as follows. If $q_{i-1,0} = 0$ then let

$$q_i(x) = q_{i-1}(x)/x \tag{2.29}$$
$$p_i(x) = p_{i-1}(x)/x \tag{2.30}$$
$$d_i = d_{i-1} + 1 \tag{2.31}$$
$$z_i = z_{i-1} \tag{2.32}$$

If $q_{i-1,0} \neq 0$ then let

$$q_i(x) = \left(q_{i-1}(x) + (q_{i-1,0}/q_{z_{i-1},0})q_{z_{i-1}}(x)\right)/x \tag{2.33}$$

which can be truncated to have degree at most $2t - i - 1$, and let

$$p_i(x) = \left(p_{i-1}(x) + (q_{i-1,0}/q_{z_{i-1},0})p_{z_{i-1}}(x)\right)/x, \tag{2.34}$$
$$d_i = 1 + \min\{d_{i-1}, d_{z_i-1}\}, \tag{2.35}$$
$$z_i = \begin{cases} i-1 \text{ if } d_{i-1} \geq d_{z_i-1} \\ z_i - 1 \text{ otherwise} \end{cases} \tag{2.36}$$

If $e \leq t$ errors have occurred during transmission then $p_{2t}(x)$ has degree e; the error locator polynomial is

$$\sigma(x) = p_{2t,e} + p_{2t,e-1}x + \ldots + p_{2t,1}x^{e-1} + x^e \tag{2.37}$$

and has e distinct roots (notice that $\sigma(x)$ is the "reverse" of $p_{2t}(x)$).

Example 2.4 *For a 3-error-correcting Reed-Solomon code, the generator polynomial is*

$$g(x) = (x+1)(x+\alpha)(x+\alpha^2)\ldots(x+\alpha^5) \tag{2.38}$$

using Table 2.3 we can obtain:

$$g(x) = 1 + \alpha^4 x + \alpha^2 x^2 + \alpha x^3 + \alpha^{12} x^4 + \alpha^9 x^5 + x^6. \tag{2.39}$$

Since $t = 3$*, we get:*

$n - k = 2t = 6$*, where* $n = 2^4 - 1 = 15$ *and* $k = n - 2t = 9$.

According to the generator polynomial, we can get $m = -1$.

Suppose the received word is $1\alpha^4 0\alpha 0\alpha^9 100000000$*, we can get the polynomial:*

$$w(x) = 1 + \alpha^4 x + \alpha x^3 + \alpha^9 x^5 + x^6. \tag{2.40}$$

1) We get the syndrome polynomials as:

$$s_0 = w(\alpha^0) = 1 + \alpha^4 + \alpha + \alpha^9 + 1 = \alpha^7, \tag{2.41}$$
$$s_1 = w(\alpha^1) = 1 + \alpha^5 + \alpha^4 + \alpha^{14} + \alpha^6 = 1, \tag{2.42}$$
$$s_2 = w(\alpha^2) = 1 + \alpha^6 + \alpha^7 + \alpha^{19} + \alpha^{12} = \alpha^9, \tag{2.43}$$
$$s_3 = w(\alpha^3) = 1 + \alpha^7 + \alpha^{10} + \alpha^{24} + \alpha^{18} = \alpha^{12}, \tag{2.44}$$
$$s_4 = w(\alpha^4) = 1 + \alpha^8 + \alpha^{13} + \alpha^{29} + \alpha^{24} = \alpha^9, \tag{2.45}$$
$$s_5 = w(\alpha^5) = 1 + \alpha^9 + \alpha^{16} + \alpha^{34} + \alpha^{30} = \alpha^7 \tag{2.46}$$

2)

$$q_{-1}(x) = 1 + \alpha^7 x + x^2 + \alpha^9 x^3 + \alpha^{12} x^4 + \alpha^9 x^5 + \alpha^7 x^6, \tag{2.47}$$
$$q_0(x) = \alpha^7 + x + \alpha^9 x^2 + \alpha^{12} x^3 + \alpha^9 x^4 + \alpha^7 x^5, \tag{2.48}$$
$$p_{-1}(x) = x^7, \tag{2.49}$$
$$p_0(x) = x^6, \tag{2.50}$$
$$d_{-1} = -1, d_0 = 0, z_0 = -1. \tag{2.51}$$

3) Preceding the step 3, we can get the following table:

Finally, we obtain:

$$\sigma(x) = \alpha^6 + \alpha^{10} x + x^2. \tag{2.52}$$

Table 2.3: Calculation results of step 3 in Example 2.4

i	$q_i - p_i$									d_i	z_i
-1	α^0	α^7	α^0	α^9	α^{12}	α^9	α^7	–	α^0	-1	
0	α^7	α^0	α^9	α^{12}	α^9	α^7	–	α^0		0	-1
1	α^3	α^0	α^{13}	α^{14}	α^{14}	–	α^0	α^7		0	0
2	α^{12}	α^7	α^6	α^{12}	–	α^0	α^8			1	1
3	α^0	α^{10}	α^9	–	α^0	α^{12}	α^1			1	2
4	0	0	–	α^0	α^{10}	α^6				2	3
5	0	–	α^0	α^{10}	α^6					3	3
6	–	α^0	α^{10}	α^6						4	3

4) According to $\sigma(x) = \alpha^6 + \alpha^{10}x + x^2 = (\alpha^2 + x)(\alpha^4 + x)$*, we get the error location numbers as* α^2 *and* α^4*.*

5) Solve the following function

$$\begin{pmatrix} 1 & 1 \\ \alpha^2 & \alpha^4 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \alpha^7 \\ 1 \end{pmatrix} \tag{2.53}$$

and we get $b_1 = \alpha^2$ *and* $b_2 = \alpha^{12}$ *.*

The most likely error pattern is:

$$e = 00\alpha^2 0\alpha^{12}0000000000. \tag{2.54}$$

Then

$$c = w + e = 1\alpha^4\alpha^2\alpha\alpha^{12}\alpha^9 100000000. \tag{2.55}$$

2.6 Low-Density Parity-Check Codes

In 1962, Gallager introduced a class of linear block codes, named low-density parity-check (LDPC) codes [3]. In 1981, Tanner generalized LDPC codes and introduced Tanner graphs to represent LDPC in [4], based on which the belief propagation (BP) algorithm is found to be a good candidate decoding algorithm. In 1995, MacKay re-invented LDPC codes [5] and pointed out that LDPC codes can provide close-to Shannon limit performances.

Let $\mathcal{F}_2^k$ denote a vector space including all binary k-tuples. This k-length vector space

can be spanned into a n-length code $\mathcal{C}$ by

$$\mathbf{c} = \mathbf{uG}, \tag{2.56}$$

where $c \in \mathcal{C}$ and $u \in \mathcal{F}_2^k$. Here $\mathbf{G}$ is a $k \times n$ generator matrix.

If there exists a matrix such as

$$\mathbf{GH}^T = \mathbf{0}, \tag{2.57}$$

$\mathbf{H}$ is called the parity-check matrix.

If $\mathbf{G}$ is a $k \times n$ matrix, H is an $(n-k) \times n$ matrix. A low-density parity-check code is a linear code with the parity-check matrix H having a low density of ones. For a regular LDPC code, we have w_c ones in each column and w_r '1' in each row. We have

$$\frac{w_r}{w_c} = \frac{n}{n-k}, \tag{2.58}$$

and $w_r \ll n$. For an irregular LDPC code, the number of ones in each column and in each row is not constant.

There are a number of constructions of LDPC codes. The most important LDPC codes include Gallager codes, MacKay codes, irregular LDPC codes, finite geometry codes, repeat-accumulate codes and array codes. A brief introduction to each of these codes can be found in [6]. We will only choose a class of finite geometry codes, in the next section, to present the construction.

One type of LDPC codes used in our simulations is type-I Euclidean geometry (EG) codes. A brief introduction to this type of LDPC codes will be presented in this section. More details can be found in [7].

We consider an m-dimensional Euclidean geometry over GF(2^s). There are 2^{ms} points in this geometry, and each point can be represented by a m-tuple over $GF(2^s)$. Therefore, the number of lines in this geometry is

$$J = \frac{2^{(m-1)s}(2^{ms}-1)}{2^s - 1}, \tag{2.59}$$

where each line has 2^s points.

Each point in this geometry is intersected by

$$\gamma = \frac{2^{ms} - 1}{2^s - 1} \tag{2.60}$$

lines.

Two lines in this geometry are either parallel to or intersected at one point.

The minimum Hamming distance of a type-I (0, s)-th order EG-LDPC code is lower bounded by

$$d_{\min} \geq \gamma + 1. \tag{2.61}$$

Therefore, m = 2-dimentional type-I (0, s)-th order cyclic EG-LDPC also has the properties as follows

$$n = 2^{2s} - 1, \tag{2.62}$$

$$n - k = 3^s - 1, \tag{2.63}$$

$$d_{\min} = 2^s + 1, \tag{2.64}$$

and

$$w_c = w_r = 2^s. \tag{2.65}$$

Finding the incident vector of a line is the key to construct a parity-check matrix, since the parity-check matrix is composed of an incident vector and all its cyclic-shifts. Let $\mathbf{I}_0^i$ denote the incident vector of length n and all its possible cyclic-shifts, where $0 \leq i \leq n-1$ and the superscript denotes the index (number) of the cyclic-shift. It is evident that $\mathbf{I}_0^0$ is the incident vector. The parity-check matrix is

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_0^0 \\ \mathbf{I}_0^1 \\ \vdots \\ \mathbf{I}_0^{n-1} \end{bmatrix}. \tag{2.66}$$

In [7], the foundations for the study of EG-LDPC codes are provided. Here we only present an algorithm to derive the incident vector.

Let GF(p^m) denote a Galois field generated by the primitive polynomial $g(x)$ under the multiplication operation $\cdot$ and the addition operation $+$. Here p denotes a prime, and m denotes a positive integer. Let α denote a primitive element which is the root of $g(x) = 0$. In the following algorithm, we use the power representation for the elements of GF(p^m). Let $\{a_i\}$, where $-1 \leq i \leq q^m - 2$ denote these p^m elements such as

$$a_i = \begin{cases} 0, & \text{for } i = -1, \\ \alpha^i, & \text{for } 0 \leq i \leq q^m - 2. \end{cases}$$

Algorithm 2.1 *Set the incident vector* $\mathbf{I}_0 = I_0 \dots I_{q^m-2} = 00 \dots 0$. *Set* $i = -1$.

Iteration:

1) If $i > q^m - 2$ *exit.*

2) $x = a_i$.

3) Set $I_y = 1$ *such as* $\alpha^y = a_{p^m-2} + a_1 \cdot x$.

4) $i = i + 1$.

5) Go to Step 1.

The systematic construction and guaranteed error correcting capability are two advantages of using geometry LDPC codes. As compared to randomly generated LDPC codes, geometry LDPC codes are not limited by the error floor and have a related simplified encoding structure. However, as we show earlier, this type of LDPC codes is not flexible in term of rate and length. Additionally, the parity-check matrix of geometry LDPC codes has the size of $n \times n$, which is larger than conventional $(n - k) \times n$. Here n is the length of the codeword and k is the length of the information.

2.7 The Tanner Graph and the Belief Propagation Decoding Algorithm

It is worthy introducing the Tanner graph before presenting the BP algorithm. A graph is composed of a vertex set V and all directed edges. If there exist V_1 and V_2 such that

$V_1 \cup V_2 = V$ and all edges having one vertex from V_1 and the other from V_2, this graph is called a Tanner graph.

For an $(n,\ k)$ linear block code $\mathcal{C}$ with parity-check matrix

$$H = \begin{bmatrix} h_{0,0} & h_{0,1} & \dots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \dots & h_{1,n-1} \\ \vdots & \vdots & h_{i,j} & \vdots \\ h_{n-k,0} & h_{n-k,1} & \dots & h_{n-k,n-1} \end{bmatrix},$$

it can be presented as a Tanner graph, in which, there are $n-k$ check nodes and n symbol nodes. If $h_{i,j} = 1$, there exists an edge connecting the symbol node j and the check node i.

For a regular LDPC code, each check node has the valency w_r, and each symbol node has the valency w_c.

The *a posteriori* probability (APP) of the symbol c_i, if we receive $\mathbf{y} = y_0 y_1 \dots y_{n-1}$, is defined as

$$\Pr(c_i = 1|\mathbf{y})) \tag{2.67}$$

The likelihood ratio (LR) $l(c_i)$ of the symbol c_i, is as

$$l(c_i) := \frac{\Pr(c_i = 0|\mathbf{y})}{\Pr(c_i = 1|\mathbf{y})}. \tag{2.68}$$

The log-likelihood ratio (LLR) is defined as

$$\log \frac{\Pr(c_i = 0|\mathbf{y})}{\Pr(c_i = 1|\mathbf{y})}. \tag{2.69}$$

Note that, in the BP algorithm for decoding LDPC codes, APP is not the only message that can be passed in the graph. Instead of APP, LR or LLR also can be considered as the passing message.

Now we present the belief propagation algorithm. Let $q_{i,j}$ and $r_{i,j}$ denote the messages passing between check nodes and symbols nodes. As shown in Fig. 2.1 and in Fig. 2.2, the symbol node is presented as a round node, and the check node is presented as a square node. The message $q_{i,j}$ is passed from the symbol node i to the check node j, and the

message $r_{i,j}$ is passed from the check node j to the symbol node i. Let S_i denote the event that the checks involving c_i symbol all are satisfied. Let T_j denote the event that the check at the node j is satisfied.

Now we can define $q_{i,j}$ as follows:

$$q_{i,j}(0) := \Pr(c_i = 0 | y_i, S_i, \{r_{i,\backslash j}\}) \tag{2.70}$$

and

$$q_{i,j}(1) := \Pr(c_i = 1 | y_i, S_i, \{r_{i,\backslash j}\}), \tag{2.71}$$

where $\{r_{i,\backslash j}\}$ denotes all messages from the check node except the check node j.

The message $r_{i,j}$ from the check node can be defined as

$$r_{i,j}(0) := \Pr(T_i | c_i = 0, \{q_{\backslash i,j}\} \tag{2.72}$$

and

$$r_{i,j}(1) := \Pr(T_i | c_i = 1, \{q_{\backslash i,j}\} \tag{2.73}$$

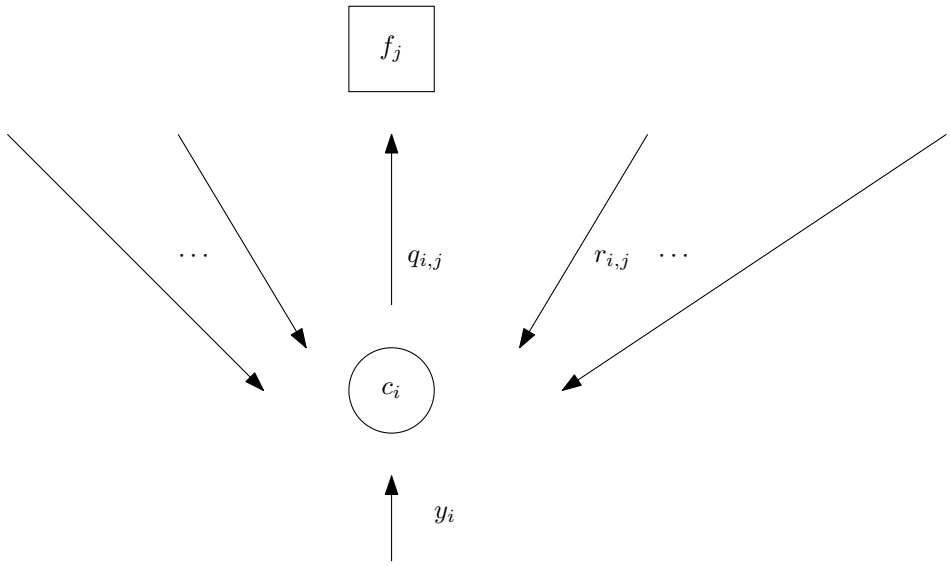


Figure 2.1: Tanner graph of the check node of a LDPC code

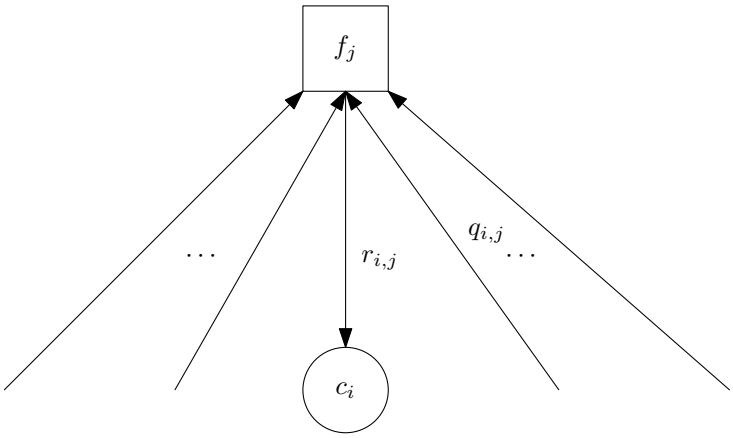


Figure 2.2: Tanner graph of the symbol node of a LDPC code

We can further derive and obtain:

$$\begin{aligned} q_{i,j}(0) &= \Pr(c_i = 0 | y_i, S_i, \{r_{i,\backslash j}\}) \\ &= \frac{\Pr(S_i | y_i, c_i = 0, \{r_{i,\backslash j}\}) \Pr(c_i = 0, y_i, \{r_{i,\backslash j}\})}{\Pr(y_i, S_i, \{r_{i,\backslash j}\})} \\ &= \frac{\Pr(S_i | y_i, c_i = 0, \{r_{i,\backslash j}\}) \Pr(c_i = 0 | y_i, \{r_{i,\backslash j}\}) \Pr(y_i, \{r_{i,\backslash j}\})}{\Pr(S_i | y_i, \{r_{i,\backslash j}\}) \Pr(y_i, \{r_{i,\backslash j}\})} \\ &= \frac{\Pr(S_i | y_i, c_i = 0, \{r_{i,\backslash j}\}) \Pr(c_i = 0 | y_i, \{r_{i,\backslash j}\})}{\Pr(S_i | y_i, \{r_{i,\backslash j}\})} \end{aligned} \tag{2.74}$$

Let P_i denote $\Pr(c_i = 1 | y_i)$. We have $\Pr(c_i = 0 | y_i) = 1 - P_i$. Since c_i is independent to $\{r_{i,\backslash j}\}$, and S_i is independent to y_i and $\{r_{i,\backslash j}\}$. Therefore, we have

$$q_{i,j}(0) = \frac{(1 - P_i) \Pr(S_i | c_i = 0, y_i, \{r_{i,\backslash j}\})}{\Pr(S_i)}. \tag{2.75}$$

We further can obtain

$$q_{i,j}(1) = \frac{P_i \Pr(S_i | c_i = 1, y_i, \{r_{i,\backslash j}\})}{\Pr(S_i)}. \tag{2.76}$$

By observation, we can find that $\Pr(S_i | y_i, c_i = 0, \{r_{i,\backslash j}\})$ is determined $\{r_{i,\backslash j}\}$. We have

$$q_{i,j}(0) = K_{i,j}(1 - P_i) \prod r_{i,\backslash j}(0), \tag{2.77}$$

and

$$q_{i,j}(1) = K_{i,j} P_i \prod r_{i,\backslash j}(1), \tag{2.78}$$

where $K_{i,j}$ is used to normalize $q_{i,j}$, since

$$q_{i,j}(0) + q_{i,j}(1) = 1. \tag{2.79}$$

We also can have

$$r_{i,\backslash j}(0) = \frac{\prod(1 - 2q_{\backslash i,j}(1))}{2}. \tag{2.80}$$

For a certain channel, $\Pr(c_i = 0|y_i) = 1 - P_i$ is given. Therefore, by (2.77), (2.78) and (2.80), we can create an iterative belief propagation algorithm. The stopping condition for this algorithm is

$$\widehat{\mathbf{c}} H^T = 0, \tag{2.81}$$

or the iteration times is more than the maximum times. Here $\widehat{\mathbf{c}}$ is obtained by

$$\widehat{c_i} = \begin{cases} 1, & \text{if } K_i P_i \prod r_{i,j}(1) > K_i(1 - P_i) \prod r_{i,j}(0), \\ 0, & \text{else,} \end{cases} \tag{2.82}$$

where K_i is the normalizing factor to hold

$$K_i P_i \prod r_{i,j}(1) + K_i(1 - P_i) \prod r_{i,j}(0) = 1. \tag{2.83}$$

2.8 How to Create Very-Long Length LDPC Codes

In this section, we will show some details of the m = 2-dimensional type-I (0, s)-th order cyclic EG-LDPC. Given s, the specifications of this types of codes are shown as in Table 2.4

Furthermore, a (65520, 61425)-LDPC code can be derived from an $s = 6$ (4095, 3367)-LDPC code following the procedure:

1) Split each column (a matrix) into 16 columns such as $n = 16 \times 4095$.

Table 2.4: EG(2, 0, s) LDPC code specifications

s	n	k
2	15	7
3	63	37
4	255	175
5	1023	781
6	4095	3367
7	16383	14197

2) Rotate each column of the parity matrix of the (4095, 3367)-LDPC code into 16 new columns.

The rotation rule is specified as follows. Let $I(i)$ denote the index of the i-th ones in the column. Therefore, we have $1 \leq I(1) < I(2) < \cdots < I(\gamma - 1) < I(\gamma) \leq n$. Let q denote the splitting depth. We have

$$\gamma = q\gamma_{ext} + r, \tag{2.84}$$

where $1 \leq r \leq \gamma_{ext} - 1$. Here γ_{ext} denotes the column weight of the parity-check matrix of the extended code. Let $I(i, j)$ denote the index of the j-th ones in the i-th rotated column. We have

$$I(i, j) = I(i + (j - 1)q), \tag{2.85}$$

where $1 \leq i \leq q$. The first r rotated columns have weight $\gamma_{ext} + 1$, in other words, $1 \leq j \leq \gamma_{ext} + 1$, and the others have weight γ_{ext}.

The above procedure, named as column splitting, can be used to create long LDPC codes from relatively short LDPC codes.

Example 2.5 *The original column is 010001011000000. We split it into 2 columns. If the first column contains the first one, then the second column contain the second one, and so on. Therefore, the original column is split into two columns such as 010000010000000 and 000001001000000.*

The incident vectors of the **A** matrices of the $m = 2$-dimentional type-I (0, s)-th order

cyclic EG-LDPC for s from two to seven are shown in Table 2.5. The notation is in left-aligned hexadecimal notation. Note that let **A** denote non-systematic generator matrix to be different from the systematic generator matrix notation **G**.

Table 2.5: Incident vectors of the **A** matrices of the EG(2, 0, s) LDPC code

s	**A** Matrix
2	0854
3	00805010000010c4
4	4100204000402008000000800000000000000c00404000000000000004011000004
5	00400000000080000000006200200000080012000000000000000000100000000
	01000000200000000200002000000000000000000000000000000800000000010
	00000000000000000400800000200000000000000000000800000004000000100 0
	82800001800000000000000008000020800000000000000010000000000000004
6	00000000080000000400000400000000000000080000000000000040400000000
	00400000000200004000000000000000000040000000000000001000000000000
	10000000000000000200000000000040000000000000000000000000000021008
	10000000000000000000001000000000000000400000000001000000000000000
	00000000000000000800000001000000000000000000000000000001200000000
	00000000000000000100000000200000000000000000000000004000000000000
	00000000000000010000000000000000800000000000000000000000000000000
	00000004010000004108000200002000000000000800000000000000000100000
	00000000000000000000000000000001000000000000000000001020000000000
	00000000000000000000000000000002000000000000000000040000000000000
	00000100000000000000000000000000000000800000000000000000000000000
	02010400000080000000000000000000000000000000000000010000000000000
	00000004000000000000000000000000000000000200000000000100008000000
	00000000000000000000000000000000000000000001000000020000000000000
	00000000000000000400000000000000000000000000000000000000000000000
	00000000000000000000000000000001001000000000000011000040000004

3 Convolutional Codes and Viterbi Decoding

3.1 Introduction

Convolutional codes are often known as nonblock linear codes, although they also have a block codeword structure over infinite fields. For a binary convolutional code, the memory and linear encoder operations are defined on GF(2). With a logic circuit implementation, the encoders consist of binary shift registers, exclusive OR and binary multiplication operations. An encoder maps an input vector of k bits to the output vector of n bits, where $n > k$. A convolutional code with the number of memory elements m can be denoted as an (n, k, m) code.

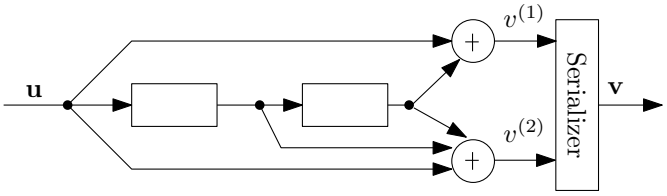


Figure 3.1: Implementation of a (2,1) convolutional encoder

An (n, k) binary convolutional code is designed to accept k-tuple binary information at the input and to generate n-tuple binary codes at the output. Since shift registers (delay elements) are used, convolutional codes are linear devices with memory. The status of the shift registers is named the state of the encoder. As shown in Fig. 3.1,**u** is the information bit shifted into the register from the left one bit at a time, and two encoded bits $v^{(1)}$, $v^{(2)}$ are generated from the XOR-adders. In the (2, 1) encoder shown in Fig. 3.1, the information sequence

$$\mathbf{u} = u_0 u_1 \dots u_t \dots \tag{3.1}$$

is encoded into

$$\begin{cases} \mathbf{v}^{(1)} = v_0^{(1)} v_1^{(1)} v_2^{(1)} \dots \\ \mathbf{v}^{(2)} = v_0^{(2)} v_1^{(2)} v_2^{(2)} \dots \end{cases} \tag{3.2}$$

and the encoded sequence is serialized as

$$\mathbf{v} = v_0^{(1)} v_0^{(2)} v_1^{(1)} v_1^{(2)} v_2^{(1)} v_2^{(2)} \dots \tag{3.3}$$

At interval t, the input of the encoder is u_t. Meanwhile, input u_{t-1} and u_{t-2} are still stored in the delay elements in the encoder. Therefore, the output of this encoder can be described as follows:

$$\begin{cases} v_t^{(1)} = u_t + u_{t-2} \\ v_t^{(2)} = u_t + u_{t-1} + u_{t-2}. \end{cases} \tag{3.4}$$

3.2 Representations of Convolutional Encoders

In a $(n,\ k)$ binary convolutional encoder (without feedback) with m delay memories

$$v_t = f(u_t, u_{t-1}, \cdots, u_{t-m}). \tag{3.5}$$

Since the encoder is a linear system, (3.5) can be written as

$$v_t = u_t G_0 + u_{t-1} G_1 + \cdots + u_{t-m} G_m \tag{3.6}$$

A general convolutional encoder can be illustrated in Fig. 3.2.

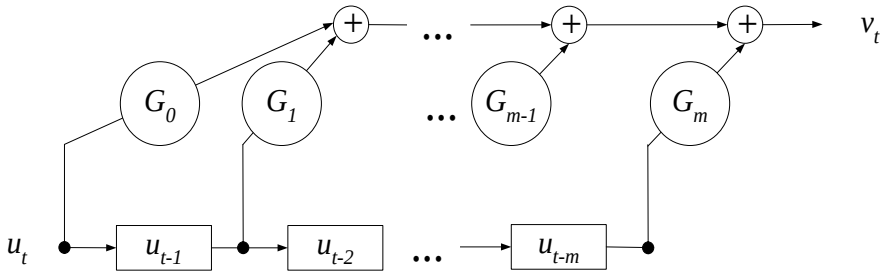


Figure 3.2: General convolutional encoder

Note that (3.6) in shorter notation becomes

$$\mathbf{v} = \mathbf{uG}, \tag{3.7}$$

where

$$\begin{pmatrix} G_0 & G_1 & \dots & G_m & & & & \\ & G_0 & G_1 & \dots & G_m & & & \\ & & G_0 & G_1 & \dots & G_m & & \\ & & & G_0 & G_1 & \dots & G_m & \\ & & & & \vdots & \vdots & \vdots & \vdots \\ & & & & & \vdots & \vdots & \vdots \end{pmatrix}$$

It is called a semi-infinite representation of a convolutional code.

A rate-k/n convolutional encoder is represented by nk generator sequences

$$\mathbf{g}_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \dots, g_{i,m}^{(j)}), \tag{3.8}$$

where $i = 1, 2, \dots, k$ and $j = 1, 2, \dots, n$.

The convolutional encoding operation can be expressed as

$$v^{(j)} = \sum_{i=1}^{k} u^{(i)} * \mathbf{g}_i^{(j)}, j = 1, 2, \dots, n. \tag{3.9}$$

Example 3.1 *For instance, the encoder in* (3.4) *has generator sequences*

$$\begin{cases} \mathbf{g}^{(1)} = (101) \\ \mathbf{g}^{(2)} = (111) \end{cases} \tag{3.10}$$

and the composite generator sequence is

$$\mathbf{g} = 11\ 01\ 11. \tag{3.11}$$

The generator sequences $\mathbf{g}_i^{(j)}$ can be presented as polynomials of finite degree in the

delay operation D as

$$\mathbf{g}_i^{(j)} = g_{i,0}^{(j)} + g_{i,1}^{(j)} D + \cdots + g_{i,m}^{(j)} D^m. \tag{3.12}$$

The generator sequences in (3.10) can be presented with polynomial as

$$\begin{cases} \mathbf{g}^{(1)}(D) = 1 + D^2 \\ \mathbf{g}^{(2)}(D) = 1 + D + D^2 \end{cases} \tag{3.13}$$

An (n, k, m) encoder can be represented by a $k \times n$ matrix $G(D)$, called the polynomial generator matrix and given by

$$\begin{pmatrix} \mathbf{g}_1^{(1)}(D) & \mathbf{g}_1^{(2)}(D) & \dots & \mathbf{g}_1^{(n)}(D) \\ \mathbf{g}_2^{(1)}(D) & \mathbf{g}_2^{(2)}(D) & \dots & \mathbf{g}_2^{(n)}(D) \\ \vdots & \vdots & \vdots & \\ \mathbf{g}_k^{(1)}(D) & \mathbf{g}_k^{(2)}(D) & \dots & \mathbf{g}_k^{(n)}(D) \end{pmatrix}$$

in which each entry is a polynomial.

Example 3.2 *For* (3.13)*, the polynomial generator matrix is*

$$G(D) = [1 + D^2 \ \ 1 + D + D^2] \tag{3.14}$$

3.3 Basic Definitions for Convolutional Codes

Definition 3.1 *The constraint length for the i'th input is*

$$v_i = \max_{1 \le j \le n} [deg(\mathbf{g}_i^{(j)}(D))] \tag{3.15}$$

Definition 3.2 *The overall constraint length is*

$$v = \sum_{i=1}^{k} v_i \tag{3.16}$$

Definition 3.3 *The memory order is*

$$m = \max_{1 \le j \le k} v_i \tag{3.17}$$

Definition 3.4 *The output constraint length is*

$$n_A = n(m+1) \tag{3.18}$$

Definition 3.5 *The input constraint length is*

$$K = v + k \tag{3.19}$$

3.4 Graphical Representation of Convolutional Codes

A convolutional code can be represented as a code tree. The code tree generated by the encoder (3.13) can be shown as in Fig. 3.3.

Also, a convolutional code can be represented by either a state diagram or a trellis diagram. Fig. 3.4 is the trellis diagram of a convolutional code in one interval. Fig. 3.5 is a typical state diagram of a convolutional code. The encoder shown in these two diagrams is similar to that of Fig. 3.3.

Graphical representations of convolutional codes not only improve the understanding of the encoding operations, but also illustrate the decoding process in a vivid way.

3.5 Distance Properties of Convolutional Codes

For a two dimensional Euclidean space, each point can be specified by a pair of real number (x, y). The Euclidean distance between the points (x_1, y_1) and (x_2, y_2) is

$$d_E\left((x_1, y_1), (x_2, y_2)\right) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.20}$$

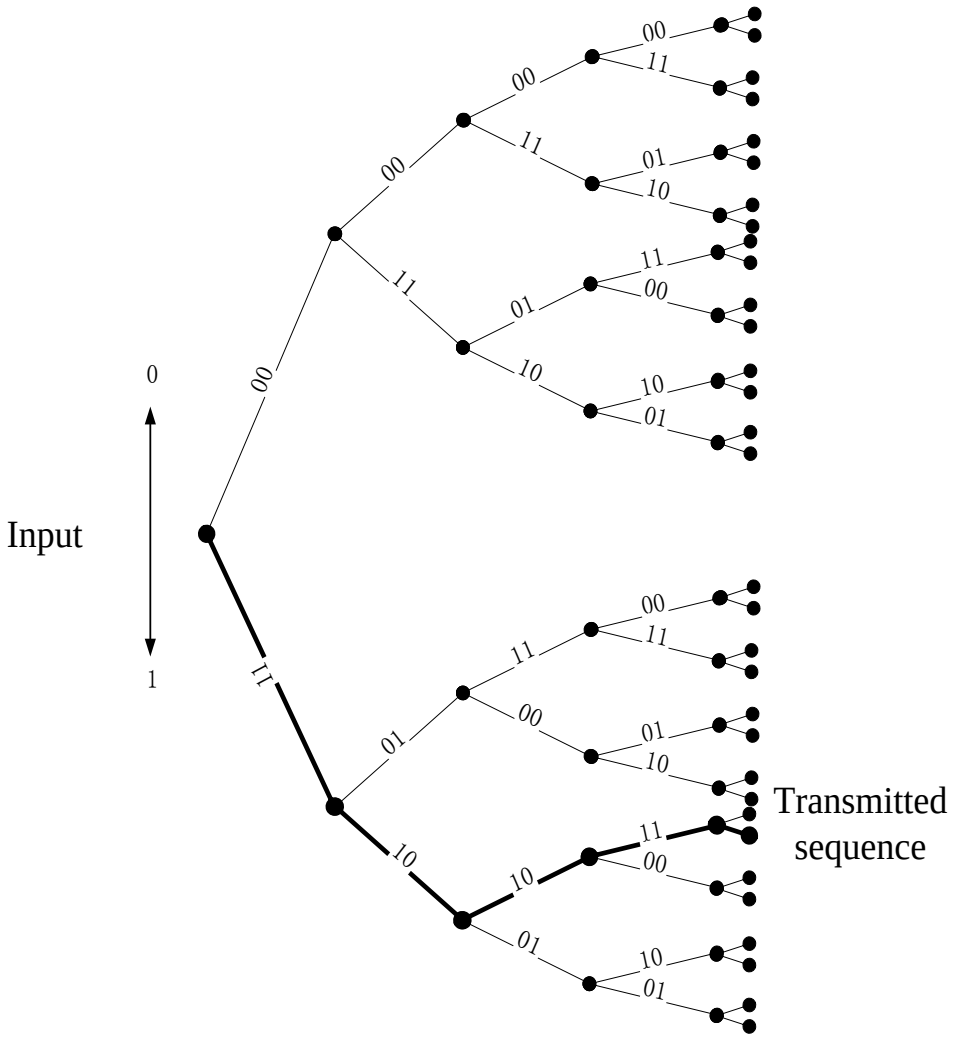


Figure 3.3: Tree structure of a convolutional code

The Euclidean distance measure has the following properties:

1) $d_E(u, v) > 0$ for $u \neq v$, and $d_E(u, v) = 0$ for $u = v$;

2) $d_E(u, v) = d_E(v, u)$;

3) $d_E(u, v) + d_E(v, w) \geq d_E(u, w)$

For a code C, the minimum distance is given by

$$d_{\min} = \min\{d_E(v_i, v_j), v_i \neq v_j \in C\} \tag{3.21}$$

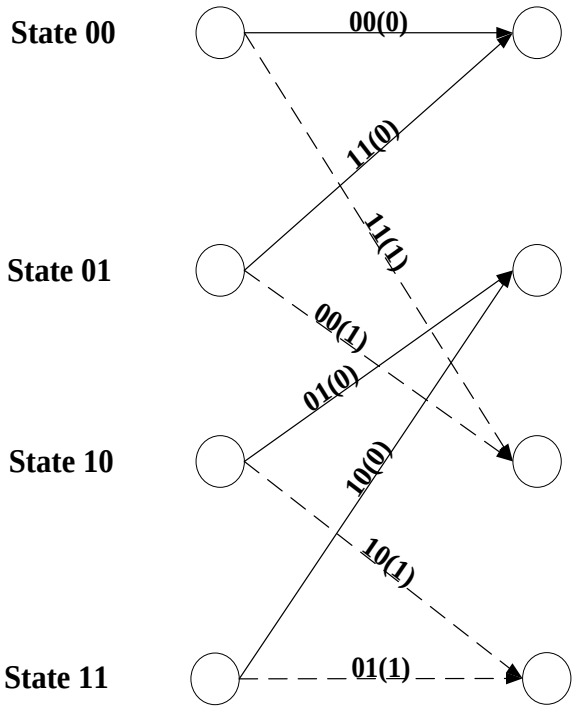


Figure 3.4: Trellis diagram of a convolutional code

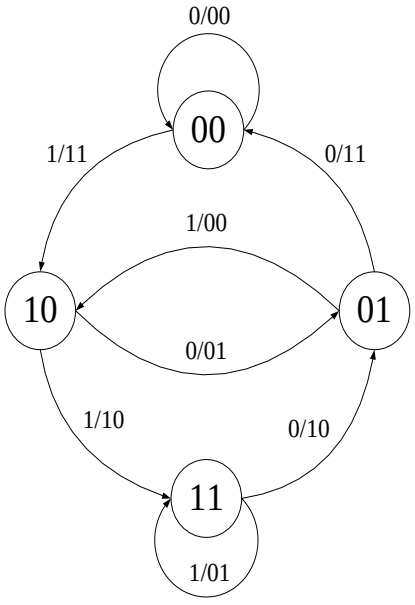


Figure 3.5: State diagram of a convolutional code

Definition 3.6 *The Hamming weight $w_H(v)$ of an n-length word $v \in GF(2^r)$ is defined*

as the number of nonzero elements in the n-tuple.

Definition 3.7 *Let n-length words v_1, $v_2 \in GF(2^r)$. The Hamming distance $d_H(v_1, v_2)$ between v_1 and v_2 is given by*

$$d_H(v_1, v_2) = w_H(v_1 - v_2). \tag{3.22}$$

Costello introduced a free distance measurement for convolutional codes.

Definition 3.8 *The free distance of a convolutional code is defined as*

$$\begin{aligned} d_{\text{free}} &= \min\{d_H(v, v') : u \neq u'\} && (3.23)\\ &= \min\{w_H(v) : u \neq 0\} && (3.24)\\ &= \min\{w_H(uG) : u \neq 0\} && (3.25) \end{aligned}$$

Remark 3.1 *The free distance is the minimum Hamming distance between any two distinct codewords generated by a convolutional encoder. The maximum error-correcting capability of a code or an encoder is determined by its free distance .*

Theorem 3.1 *The maximum error-correcting capability t_{free} of a code or an encoder is*

$$t_{\text{free}} = \left\lfloor \frac{d_{\text{free}} - 1}{2} \right\rfloor \tag{3.26}$$

Definition 3.9 *The distance spectrum of a code or an encoder is the sequence*

$$n(d_{\text{free}} + i) i = 0, 1, 2, \ldots \tag{3.27}$$

where $n(d_{\text{free}} + i)$ denotes the number of weight $d_{\text{free}} + i$ codewords that correspond to a nonzero first information block and all-zero last m information blocks.

3.6 Degenerate and Catastrophic Convolutional Code Encoder

In the above sections, some properties and analysis measurements of convolutional codes are presented. Besides these disciplines, degenerate and catastrophic properties of a convolutional code are also important conditions to judge good codes.

Definition 3.10 *A rate-*$1/n$*, constraint length* v *convolutional encoder is nondegenerate if:*

1) there is $j_1 \in \{1, 2, \ldots, n\}$ *such that* $g_0^{(j_1)} = 1$,

2) there is $j_2 \in \{1, 2, \ldots, n\}$ *such that* $g_v^{(j_2)} = 1$, *and*

3) there is $j_1, j_2 \in \{1, 2, \ldots, n\}$, $j_1 \neq j_2$ *such that* $g^{(j_1)} \neq g^{(j_2)}$.

Remark 3.2 *Nondegenerate convolutional encoders are effectively of shorter constraint length.*

Definition 3.11 *A convolutional code is catastrophic if and only if its state diagram has a zero-weight cycle other than the self-loop around the all-zero state.*

Example 3.3 *The state diagram of a (2, 1, 2) convolutional encoder with generator polynomial* $g^{(1)}(D) = 1+D$ *and* $g^{(2)}(D) = 1+D2$ *is shown in Fig. 3.6. The input sequence* $u = 111$ *can result in output sequence* $v = 11010000$*, which will make the decoder estimate an all-zero sequence. Therefore, it will make a catastrophic error.*

3.7 Punctured Codes, Rate-Compatible Encoder and Unequal Error Protection

Definition 3.12 *A punctured code is obtained by periodically deleting (or puncturing) encoded symbols from ordinary encoded sequences. This process is named puncturing.*

Definition 3.13 *If a rate-*$1/n$ *parent encoder is punctured by deleting some of the* nP *encoded bits corresponding to* P *information bits, then* P *is called the puncturing period.*

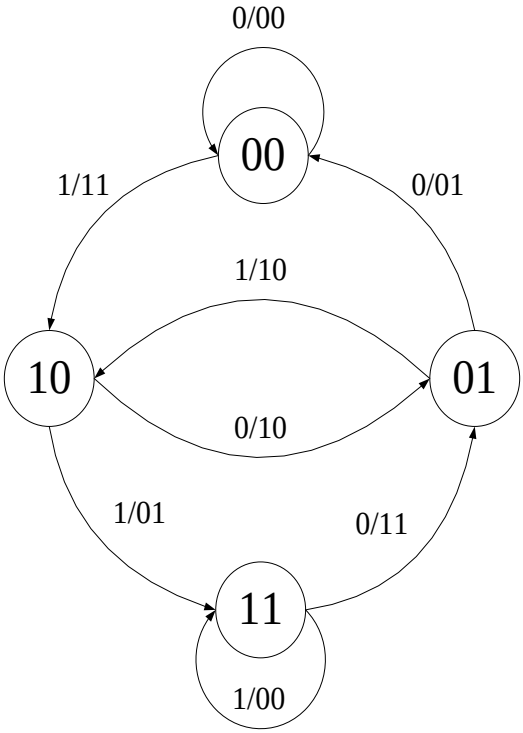


Figure 3.6: State diagram of a catastrophic convolutional code

After puncturing the symbols from the encoded sequences, the encoded sequences corresponding to the same amount of information are reduced. Therefore, the rate of the encoder is increased by the puncturing process. The ordinary convolutional codes are named the parent codes. The punctured codes generated from the parent codes are named the child codes.

Definition 3.14 *For a rate-*$1/n$ *parent encoder, the puncturing pattern can be represented as an* $n \times P$ *matrix* $\mathbf{P}$ *whose elements are 1's and 0's, with a 1 indicating inclusion and a 0 indicating deletion.*

Example 3.4 *Given an encoder with generator polynomial*

$$G = \begin{pmatrix} 1 + D^2 & 1 + D + D^2 \end{pmatrix} \tag{3.28}$$

and a puncturing matrix as

$$\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{3.29}$$

it indicates that within two encoded blocks, the first bit of the second encoded block is

deleted.

Fig.3.7 shows the puncturing process in Example 3.4, where 'x' denotes the bit deleted. Note that the solid lines represent information bit '0' and the dash lines represent information bit '1'.

The parent encoder in Example 3.4 is punctured into an $R = 2/3$ encoder, and the equivalent $R = 2/3$ encoder can be shown as in Fig.3.8. Note that the solid lines represent information bits '00', the dash lines represent information bits '01', the dash dotted lines represent information bits '10' and the dotted lines represent information bit '11'.

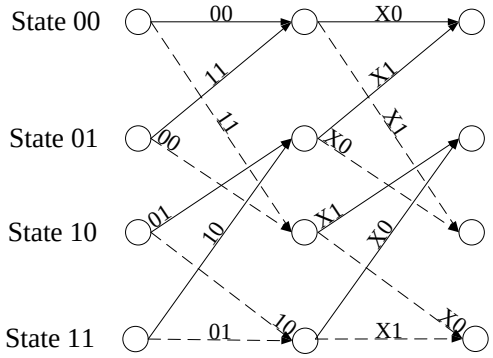


Figure 3.7: Trellis diagram of a punctured convolutional encoder

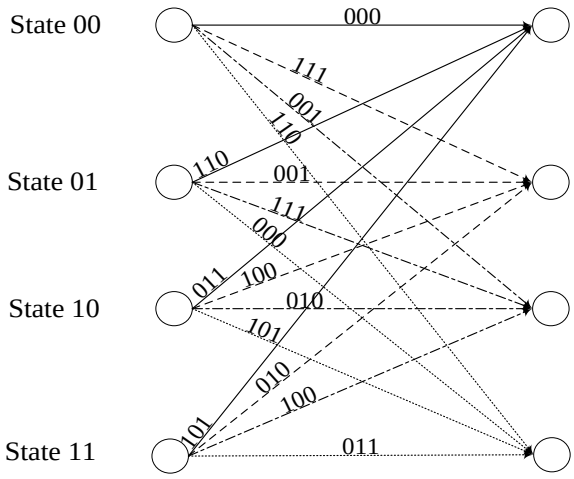


Figure 3.8: Trellis diagram of a $R = 2/3$ convolutional encoder

The puncturing process reduces the free distance of the parent code. However, the free distance of the punctured code can be comparable with that of an ordinary convolutional code having the same rate and constraint length as the final rate after puncturing.

Meanwhile, the decoding of the punctured codes is simpler than that of the ordinary convolutional code with the same rate and constraint length. That is the reason that punctured codes are widely used in practical applications.

The progress in communication technology led to a significant increase of the types and numbers of applications. Furthermore, due to a variety of protection requirements on transmission, channel coding for unequal error protection (UEP) has attracted a lot of interests.

As excellent error control codes to achieve UEP, punctured codes are widely exploited in commercial use. Despite the reduction in the free distance, punctured codes enhance the efficiency of the bandwidth use of the channel and simplicity of the decoder structure. Moreover, the free distance of punctured code is comparable with that of a conventional convolutional code having the same rate and constraint length as the final rate after puncturing.

3.8 Maximum Likelihood Decoding

In a typical digital communication or storage system as shown in Fig. 3.9, encoded information v is sent via the channels and corrupted by noise e. At the receiver, the decoder attempts to recover the words transmitted and get the most likely word sequences v', and inverse the v' to get the most likely information u'. Here v' is the most likely codeword for the case that r is received. This operation is called maximum likelihood decoding. Given channel transition probabilities below 0.5, the maximum likelihood decoding for a binary symmetric channel is equivalent to selection of the codeword which minimizes the Hamming distance between r and v.

3.9 Viterbi Algorithm

Forney showed that the Viterbi algorithm is a maximum likelihood algorithm: given a sequence r of observations of a discrete-state Markov process in memoryless noise, the algorithm finds the state sequence v' for which the a posteriori probability $P(v'|r)$ is maximum.

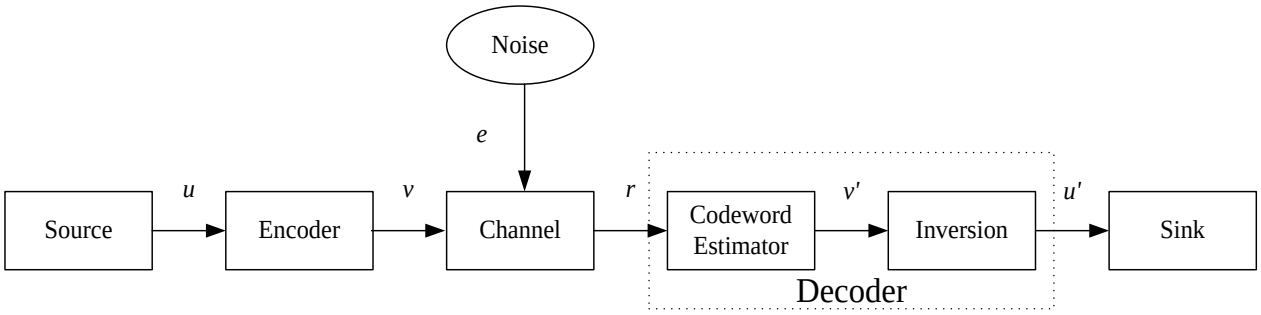


Figure 3.9: A digital communication/storage system

Let $X = \{0, 1, \ldots, B-1\}$ denote the state space of a Markov process and x_t denotes the state of this Markov process at time t, where $x_t \in X$.

For a memoryless Markov process, let v' denote the most likely sequence given r is received, we can get

$$P(x_{t+1}|x_0, x_1, \ldots, x_t) = P(x_{t+1}|x_t), \tag{3.30}$$

$$P(v'|x) = P(v'|\xi) = \prod_{t=0}^{T-1} P(v'_t|\xi_t) \tag{3.31}$$

and

$$P(x, v') = P(x)P(v'|x) \tag{3.32}$$

$$= \prod_{t=0}^{T-1} P(x_{t+1}|x_t) \prod_{t=0}^{T-1} P(v'_t|x_{t+1}, x_t), \tag{3.33}$$

where ξ_t denotes the transition from a state x_t to x_{t+1}, and ξ denotes the transition $\{\xi_0, \xi_1, \ldots, \xi_{T-1}\}$.

To achieve the maximum likelihood decoding, we need to find the state sequence that can maximize $P(x|v')$.

Let

$$\lambda(\xi_t) = -\ln P(x_{t+1}|x_t) - \ln P(v'_t|\xi_t), \tag{3.34}$$

we obtain

$$-\ln P(x_{t+1}|x_t) = \sum_{i=0}^{T-1} \lambda(\xi_i) \tag{3.35}$$

Thus, according to (3.35), we need to look for the least $\sum_{i=0}^{T-1} \lambda(\xi_i)$ to obtain the maximum $P(x|v')$.

For state x_t, the shortest path segment is called the survivor corresponding to x_t, and is denoted as $\chi(x_t)$. For B states of time instance, we need to remember $\chi(x_t)$ and their length $\Gamma(x_t) = \lambda(\chi(x_t))$.

Thus, the Viterbi algorithm can be shown as follows:

Storage:

1) $\chi(x_t), 1 \leq x_t \leq B$

2) $\Gamma(x_t), 1 \leq x_t \leq B$

where t is the time index.

Assume that the start state is x_0.

Initiation:

1) t=0

2) $\chi(x_0) = x_0$; $\chi(b)$ arbitrary, $b \neq x_0$

3) $\Gamma(x_0) = 0$; $\Gamma(b) = \infty$, $b \neq x_0$

Recursion:

Repeat the following steps until $t = T$

1) Compute $\Gamma(x_{t+1}, x_t) = \Gamma(x_t) + \lambda(\xi_t)$, for all ξ_t ;

2) $\Gamma(x_{t+1}) = \min(\Gamma(x_{t+1}, x_t))$ for each x_{t+1}; store $\Gamma(x_{t+1})$ and corresponding $\chi(x_{i+1})$.

3) Set $t = t + 1$ and go to step 1.

Result: the shortest path is the survivor $\chi(x_T)$.

Hard-decision decoding takes a stream of bits and consider definitely each bit as 0 or 1. The decision mechanism is application-dependant, *e.g.*, threshold detection.

Soft-decision decoding takes a stream of bits and consider each bit as 0 or 1 in a probabilistic sense.

We assume the threshold detection is used at the receiver of a pulse amplitude modulation (PCM) system. Bits 0 and 1 are mapped into pulses with amplitudes $-1mV$ and $1mV$ respectively, and sent over the channel. Assume the noise over the channel is Gaussian with the distribution $(0, \sigma^2)$.

For hard-decision decoding, the detector first decides an optimal threshold (in this case is $0mV$). If the received signal is above the optimal threshold, we consider a bit 1 received; if the received signal is less than the optimal threshold, we consider a bit 0 received.

For soft-decision decoding, the decoder records the differences between the received signal with either error-free amplitudes. The differences are inversely proportional to the reliabilities of the decisions.

For one codeword with multiple dimensions, Euclidean distances are calculated to make the soft-decision decoding.

Example 3.5 *The repetition code of length 3 has following 2 codewords:*

$$\mathbf{c}_1 = 000 \rightarrow (-1, -1, -1) \tag{3.36}$$

$$\mathbf{c}_2 = 111 \rightarrow (1, 1, 1) \tag{3.37}$$

If the received signal is $\mathbf{r} = (1.25, 0.91, -0.95)$*, for the soft decisions the Euclidean distance is*

$$d(\mathbf{r}, \mathbf{c}_1) = \sqrt{(1.25 + 1)^2 + (0.91 + 1)^2 + (-0.95 + 1)^2} = 2.95 \tag{3.38}$$

and

$$d(\mathbf{r}, \mathbf{c}_2) = \sqrt{(1.25 - 1)^2 + (0.91 - 1)^2 + (-0.95 - 1)^2} = 1.97 \tag{3.39}$$

A soft-decision decoder would say more likely, the codeword 111 was sent, since $d(\mathbf{r}, \mathbf{c}_2) < d(\mathbf{r}, \mathbf{c}_1)$.

Example 3.6 (hard-decision Viterbi) *Assume the message is 0111100110. The encoder can be defined by the generator matrix* $g(D) = [1 + D^2, 1 + D + D^2]$. *Then the encoded sequence 00 11 10 01 01 10 11 11 10 10 is sent over the channel. But we received 00$\underline{0}$1 10 $\underline{1}$1 01 10 11 11 10 10. The underlined symbols are disturbed by additive errors. Fig. 3.10 shows how to retrieve the message based on a disturbed received sequence by using the Viterbi algorithm.*

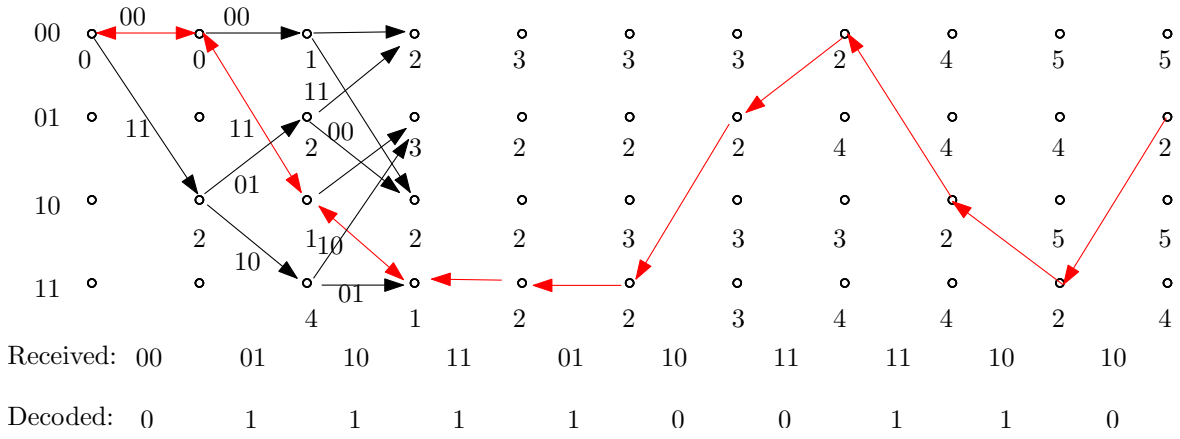


Figure 3.10: An example of hard-decision Viterbi decoding

3.10 Error Upper Bounds for Time-Invariant Convolutional codes

Consider a typical convolutional code in Fig. 3.11. The self-loop at the zero-state can be split in two: the source state and the sink state. Let $W^0 = 1$, W^1 and W^2 denote the weight of the output of the branches in the state diagram. We can obtain a signal flow diagram as shown in Fig. 3.12.

Definition 3.15 *In a typical state diagram of a convolutional code, let the input to the source (left zero state) be 1 and let* $T(W)$ *denote the generating function for the path weight* W. *We call* $T(W)$ *the path weight enumerator.*

Example 3.7 *In Fig. 3.12, the branches are labeled* $W^0 = 1$, W *or* W^2, *where the exponent corresponds to the weight of the particular branch.*

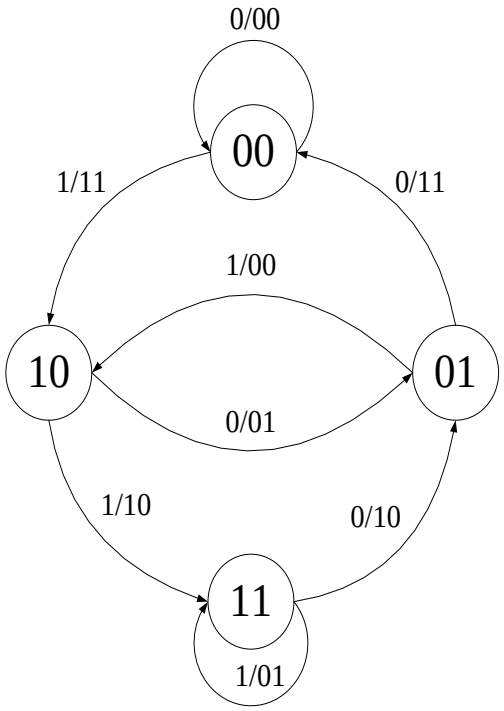


Figure 3.11: State diagram of a (2,1) convolutional encoder

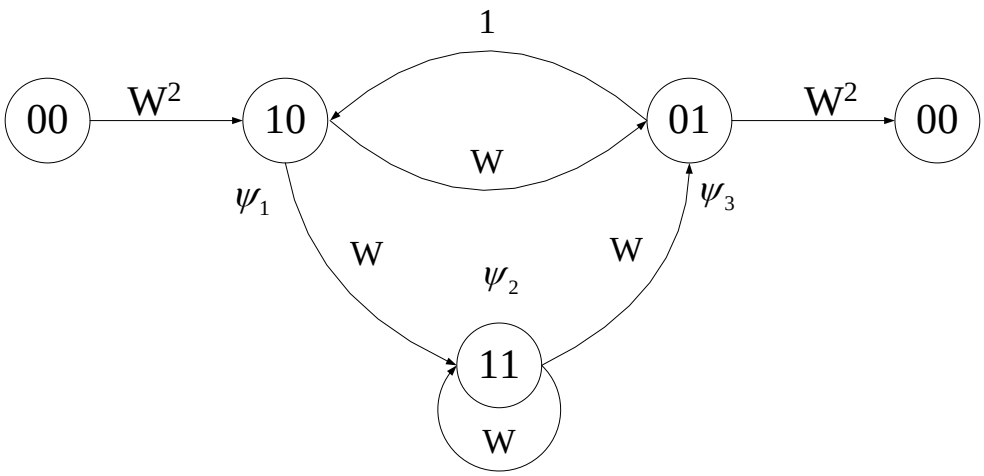


Figure 3.12: Signal flow diagram of a (2,1) convolutional encoder

Let ψ_1, ψ_2 and ψ_3 be the dummy variables representing the weights to the intermediate nodes. According to Fig. 3.12, when the input to the left zero state is 1, we can obtain:

$$\psi_1 = \psi_3 + W^2 \tag{3.40}$$

$$\psi_2 = W\psi_1 + W\psi_2 \tag{3.41}$$

$$\psi_3 = W\psi_1 + W\psi_2 \tag{3.42}$$

Thus, according to Definition 3.15 and Fig. 3.12, we can get:

$$T(W) = W^2\psi_3. \tag{3.43}$$

Combining (3.40), (3.41), (3.42) *and* (3.43) *we get*

$$T(W) = W^5/(1-2W) \tag{3.44}$$

(3.44) *can be also represented as*

$$T(W) = W^5 + 2W^6 + 4W^7 + \ldots + 2^k W^{k+5} + \ldots \tag{3.45}$$

According to (3.45)*, we find that this encoder has* $d_{\text{free}} = 5$ *and the spectral components are 1, 2, 4 ...*

3.11 Viterbi Upper Bounds of Burst Error Probability

Definition 3.16 *In the trellis diagram of a convolutional code, the burst error probability* P_B *is the probability that an error burst starts at a given node.*

Theorem 3.2 *The burst error probability when using a convolutional code for communication over the binary symmetric channel (BSC) with crossover probability* ϵ *and maximum-likelihood decoding is upper-bounded by*

$$P_B < \sum_{d=d_{\text{free}}}^{\infty} n_d \left(2\sqrt{\epsilon(1-\epsilon)}\right)^d \tag{3.46}$$

$$= T(W)|_{W=2\sqrt{\epsilon(1-\epsilon)}} \tag{3.47}$$

where $T(W)$ *is the path weight enumerator of the encoder.*

4 Interleaving and Concatenated Coding System

4.1 Interleaving

In practice, the interleaving technique is widely used to cooperate with error correcting codes to improve the burst error correcting capability. The interleaving technique is implemented by rearranging the order of symbols transmitted.

Suppose now $n\lambda$ symbols are needed to be transmitted. As shown in Fig. 4.1, these symbols can be grouped into λ segments and each segment has n symbols. In the transmission, the first symbols of each segment are first sent then the second symbols of each segment, and so on. Here λ is named the interleaving depth.

In a practical channel, it is more common that consecutive symbols are affected. This type of error is called a burst error. As shown in Fig. 4.2, where the symbols affected are marked, while the transmission is corrupted by a burst error, a long sequence of symbols becomes unreliable. However, with the interleaving technique, the long sequence of error symbols is broken into smaller pieces as shown in Fig.4.3, which should be fit to be corrected by the error correcting code.

For some specific applications, symbols are encoded or inserted in certain positions intentionally, such as the Marker codes used to indicate the boundaries of symbols. The technique to change the order of the other symbols while keeping the positions of these specific symbols is named partial interleaving. A typical partial interleaving is shown in Fig. 4.4.

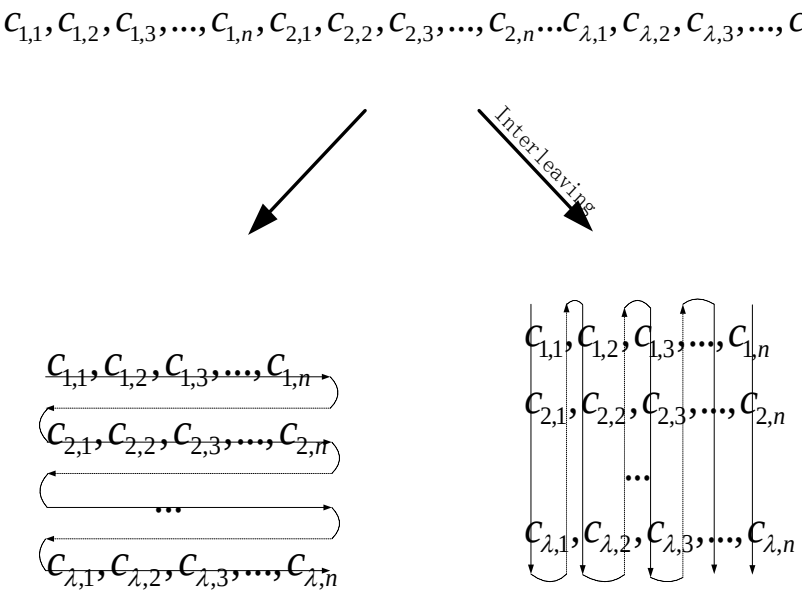


Figure 4.1: An interleaving system

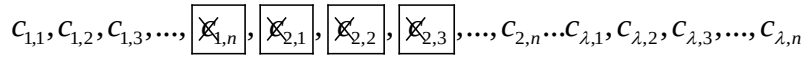


Figure 4.2: Error pattern before de-interleaving

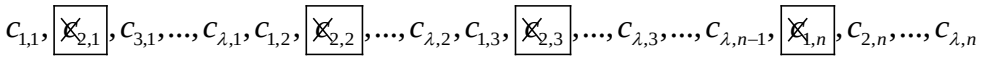


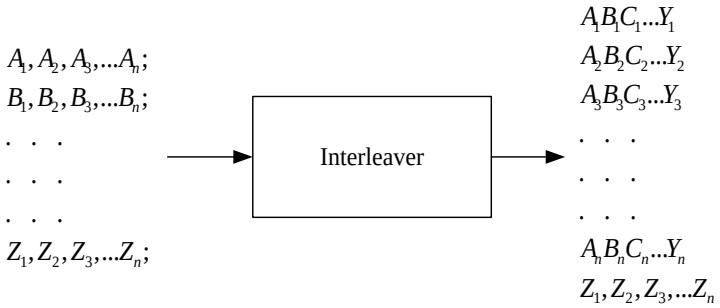
Figure 4.3: Error pattern after de-interleaving

4.2 Concatenated System

A concatenated code is one coding system that uses two levels of coding, an inner code and an outer code, to achieve the desired error performance. A concatenated coding system using a Viterbi-decoded convolutional inner code and a Reed-Solomon outer code is widely utilized in practical applications.

In this system as shown in Fig. 4.5, as the inner decoder, the Viterbi algorithm decoder can decode the soft quantized code from the demodulator. And the burst errors generated by the Viterbi decoder can be processed by the combined decoding techniques of the Reed-

Solomon decoder and de-interleaving.


$A_1, A_2, A_3, \ldots A_n;$
$B_1, B_2, B_3, \ldots B_n;$
. . .
. . .
. . .
$Z_1, Z_2, Z_3, \ldots Z_n;$
Interleaver
$A_1B_1C_1 \ldots Y_1$
$A_2B_2C_2 \ldots Y_2$
$A_3B_3C_3 \ldots Y_3$
. . .
. . .
. . .
$A_nB_nC_n \ldots Y_n$
$Z_1, Z_2, Z_3, \ldots Z_n$

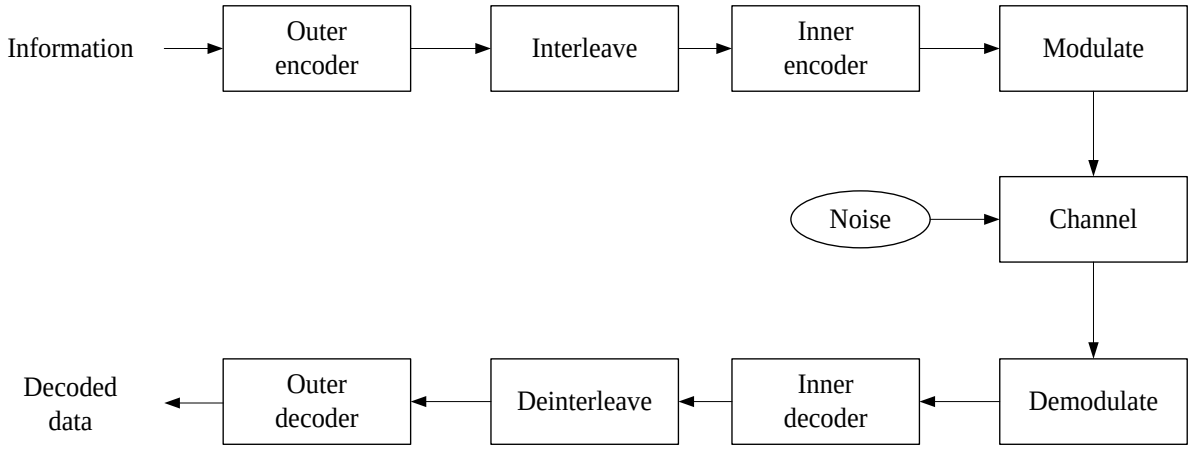


Figure 4.5: A concatenated coding system

References

[1] C. E. Shannon, "A mathematical theory of communications." *Bell Syst. Tech. Journal.*, pp. 623–656, July 1948.

[2] C. E. Shannon and W. Weaver, *The mathematical theory of communication*, 1st ed. Urbana: University of Illinois Press, 1949.

[3] R. G. Gallager, "Low-density parity-check codes." *IRE transactions on information theory*, pp. 21–28, Jan. 1962.

[4] R. M. Tanner, "A recursive approach to low complexity codes." *IEEE Trans. Inform. Theory*, pp. 533–547, Sept. 1981.

[5] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices." in *Cryptography and coding 5th IMA conference*, vol. 1025, 1995, pp. 100–111.

[6] W. E. Ryan, "An introduction to LDPC codes." *Online*, Aug. 2003.

[7] S. Lin and D. J. Costello, Jr., *Error control coding*, 2nd ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2004.