Lattice Coding Theory
Lecture Notes

Brian M. Kurkoski

2023 October 11

Lattice Coding Theory Lecture Notes
2022
Brian M. Kurkoski
kurkoski@jaist.ac.jp

Cover image: Flickr/Matthew T Rader Photography

Contents

Preface

Welcome

Lattices are error-correcting codes over the real numbers. Real numbers describe the physical world, particularly the physical media used in communications. In wireless communications for example, when two signals are transmitted at the same time they will superimpose — that is, they add, making lattice codes a natural fit for wireless communications, particularly multiuser communications. In the physical world, 1 plus 1 is 2, and it is the same for lattices.

Lattices have captured the attention of mathematicians since the late 19th century. Led by Conway and Sloane starting in the 1970s developed lattices, including numerous constructions, particularly in low dimensions with elegant algebraic structure. Throughout the 1990s, Forney, Calderbank and others pioneered the further development of lattices for communications, as well as non-lattice coded modulation. Lattices are useful in the first of their two practical communication applications, to bandwidth efficient communications, which were voice band modems, because telephone bandwidth requires the use of efficient modulation techniques.

Information theoretic results using lattices have appeared, many inspired by the work of Ram Zamir and his colleagues. These results show that lattices can achieve the capacity, or nearly so, for a variety of communication systems. Typical examples are communications scenarios such as point-to-point communications, multi-user communications, multiple-antenna communications and distributed sourcing coding. Characteristic of information theoretic results is the use of lattices which are capacity-achieving, but have little other structure.

For these information theoretic results to be realized in practice, specific constructions of finite dimension are needed. Error-correcting codes underwent a true paradigm shift with the 1993 invention of turbo codes, and the subsequent rediscovery of low-density parity check codes. Since the 2000s, various lattices versions of high dimension have been successfully developed, inheriting the strengths of turbo codes and low-density parity-check codes.

The central goal of *Lattice Coding Theory* is to bring together classical and recent lattice constructions, as many important recent results are distributed in the academic literature. The focus is on lattices for reliable communications, that is, where the lattice is being used as an error-correcting code in a system with noise. Lattices constructions are selected for their importance and promise

in future communications systems.

Reflected in the title, *Lattice Coding Theory* uses the framework of coding theory. Finite-field codes, such as Hamming codes, convolutional codes, BCH codes and LDPC codes, are well-studied and their use in practical communication systems is well established. Many lattices have direct connections with finite-field codes, particularly through Construction A and Construction D/D'. Many properties of codes extend naturally to lattices. Finite-field codes are often written with a matrix representation, and likewise matrix representations for lattices are emphasized here.

By presenting lattice coding theory using finite-field error-correcting codes, those already familiar with error-correcting codes can quickly grasp lattices. Figures of two and three-dimensional lattices can meaningfully and accurately describe important concepts. Group-theoeric structure not needed for applications will be de-emphasized in our drive to develop lattices for wireless communications. The content is not practically itself, but rather presents theory in a way that is understandable to practitioners. As with many books on coding theory, idealized communication channel models are used to illustrate the lattice properties.

Finally, lattices are inherently interesting. Part of my enthusiasm stems from the fact that lattices are elegant and both visually and mathematically pleasing. It is interesting to note that for finite-field codes, two-dimensional figures are more conceptual than accurate

Notation

While the emphasis of these lecture notes is on conceptual understanding, some mathematics is necessary. An attempt has been made to use consistent notation throughout, some of which is summarized in the table below.

$\mathbf{x}$	vectors are indicated by lower case bold face, for example: $$\mathbf{x} = [x_1, x_2]$$
$[\cdot]^{\mathrm{t}}$	vector and matrix transpose, for example: $$\mathbf{x}^{\mathrm{t}} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
$\mathbf{x} + \mathbf{y}$	vector addition
$\mathbf{G}$	matrices are indicated by upper case bold face, for example: $$\mathbf{G} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}$$
$\mathbf{I}_m$	m-by-m identity matrix, for example: $$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$\det(\mathbf{G})$	the determinant of a square matrix $\mathbf{G}$.
$\mathcal{A}$	sets are indicated by calligraphic script, for example $\mathcal{A} = \{0, 1, 2, 3, 4\}$
$\lvert\mathcal{A}\rvert$	cardinality of a set, for example $\lvert\mathcal{A}\rvert = 5$
$\mathcal{A} \setminus a$	set subtraction, for example $\mathcal{A} \setminus \{0, 2\} = \{1, 3, 4\}$
$\mathcal{G}, \mathcal{H}$	a group $\mathcal{G}$ and a subgroup $\mathcal{H}$
$\mathcal{R}$	a ring is indicated
$\mathbb{F}$	a polynomial ring
$\mathbb{F}_q$	a field of size q.
$\mathbb{Z}$	the set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{Z}^n$	the set of all integer vectors in n-dimensions, for example $(-4, 0, 1, 3) \in \mathbb{Z}^4$
$\mathbb{R}$	the set of real numbers

Λ	a lattice
$\mathcal{R}$	a shaping region in n dimensions $\mathcal{R} \subset \mathbb{R}^n$. A lattice code is $\Lambda \cap \mathcal{R}$
$\mathbb{R}^n$	the n-dimensional Euclidean space
$\|\|\mathbf{x}\|\|^2$	squared length of vector $\mathbf{x}, \sum_{i=1}^{n} x_i^2$
$\|\|\mathbf{x}-\mathbf{y}\|\|^2$	squared Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$
$\binom{n}{k}$	n choose k, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Linear Algebra

The following are properties of matrices that students should already be familiar with. Let $\mathbf{A}$, etc. be matrices:

1. Matrix transpose: $(\mathbf{AB})^{\mathrm{t}} = \mathbf{B}^{\mathrm{t}}\mathbf{A}^{\mathrm{t}}$
2. Matrix inverses: $(\mathbf{A}^{\mathrm{t}})^{-1} = (\mathbf{A}^{-1})^{\mathrm{t}}$ and $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$.
3. Orthogonal matrix $\mathbf{Q}$ satisfies $\mathbf{Q}^{\mathrm{t}}\mathbf{Q} = \mathbf{I}$. Equivalently, $\mathbf{Q}^{\mathrm{t}} = \mathbf{Q}^{-1}$.
4. If $\mathbf{A}$ has only real entries, then $\mathbf{A}^{\mathrm{t}}\mathbf{A}$ is a positive semidefinite matrix.

Chapter 1

Error Correcting Codes

This chapter gives an overview of coding theory: abstract algebra, principles of block codes, and Euclidean-space codes. These are the concepts that will be used throughout the course.

1.1 Error Correcting Code Concepts

1.1.1 Communications System

Error-correcting codes provide reliable communications over an unreliable channel. The central idea is that a transmitter wants to send a message to a receiver using a communication channel which causes errors to occur. A model of a communications system is shown in Fig. 1.1. It consists of five parts: an information source, an encoder, a channel, a decoder, and an information sink. The information source produces a message $\mathbf{u}$. An encoder transforms the message to a codeword $\mathbf{c}$, adding redundancy to the message. The codeword $\mathbf{c}$ is transmitted over an unreliable channel, meaning that errors are added to $\mathbf{c}$. The output of the channel is $\mathbf{y}$ and a decoder, with knowledge of the encoding, makes an estimate $\widehat{\mathbf{c}}$ or $\widehat{\mathbf{u}}$ from $\mathbf{y}$. This estimate is delivered to the information sink. The goal is that $\widehat{\mathbf{u}}$ should be the same as $\mathbf{u}$ (or, $\widehat{\mathbf{c}}$ should be the same as $\mathbf{c}$).

To show how error-correcting codes work, two examples are given, a repeat code and a Hamming code.

1.1.2 Example: Repeat Code

Consider the repeat-by-seven code. The information source produces $u = 0$ or $u = 1$, and encodes to $\mathbf{c}$ according to:

u	$\mathbf{c}$
0	0000000
1	1111111

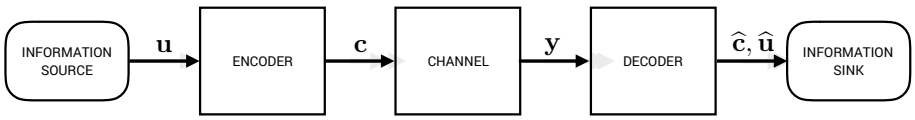


Figure 1.1: Model of a communication system with encoder, channel and decoder.

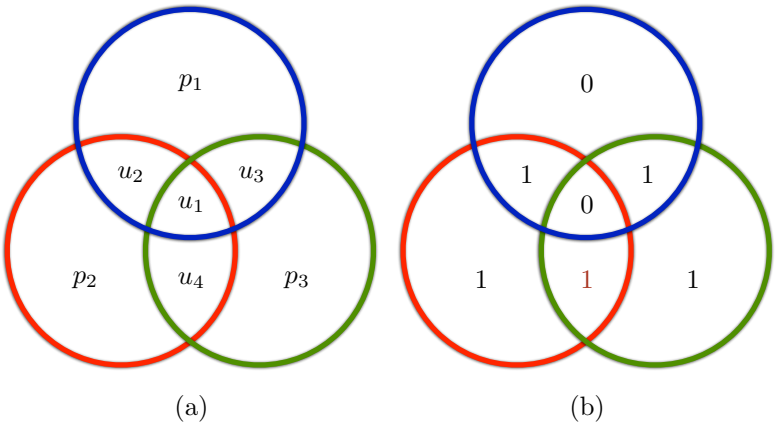


Figure 1.2: Graphical representation of a Hamming code. (a) Information bits are u_1, u_2, u_3, u_4 and parity bits are p_1, p_2, p_3. The number of bits inside each circle must be even. (b) The sequence $\mathbf{y} = [0, 1, 1, 1, 0, 1, 1]$ is received. A single error can be corrected.

Suppose $u = 1$, and the encoder transmits the codeword:

$$\mathbf{c}_1 = [1111111]. \tag{1.1}$$

The channel introduces three errors, such that the channel output is:

$$\mathbf{y} = [0110101]. \tag{1.2}$$

The decoder uses a "majority vote" decoding algorithm. Seeing that there are more 1's than 0's, the decoder chooses:

$$\widehat{\mathbf{c}} = [1111111], \tag{1.3}$$

as the estimated codeword, which corresponds to $\widehat{u} = 1$. In this case, the decoder is correct.

The repeat-by-seven code can correct any pattern of 0, 1, 2 or 3 errors. But, it used the channel seven times to transmit only one bit of information. The following code uses the channel more efficiently.

1.1.3 Example: Hamming Code

Consider the following code, called the (7,4) Hamming code. The source produces one of 16 symbols, with $\mathbf{u} = [u_1, u_2, u_3, u_4]$ where $u_i \in \{0, 1\}$ are four

binary source symbols. This will be encoded to a codeword with 7 bits:

$$\mathbf{c} = [u_1, u_2, u_3, u_4, p_1, p_2, p_3], \tag{1.4}$$

where p_1, p_2 and p_3 are called *parity bits*. A graphical representation of the code is shown in Fig. 1.2-(a), where each circle contains four bits. The seven code bits must satisfy the following condition:

> The number of 1's inside each circle must be even. Equivalently, the modulo-2 sum of the bits inside each circle must be 0.

The parity bits are selected to satisfy this condition.

Suppose $\mathbf{u} = [0, 1, 1, 0]$. Then $p_1 = 0$, $p_2 = 1$, $p_3 = 1$ makes the number of 1's inside each circle even, and the transmitted codeword is:

$$\mathbf{c} = [0, 1, 1, 0, 0, 1, 1]. \tag{1.5}$$

After transmission through the channel, one error occurs:

$$\mathbf{y} = [0, 1, 1, 1, 0, 1, 1], \tag{1.6}$$

which is illustrated in Fig. 1.2-(b). The decoder knows the encoding rule. The blue circle has an even number of ones, and the red and green circles have an odd number of ones. Since only u_4 is inside the red and green circles, but not the blue circle, the decoder changes y_4 from $1 \to 0$, so that the estimated codeword is:

$$\widehat{\mathbf{c}} = [0, 1, 1, 0, 0, 1, 1]. \tag{1.7}$$

In this case, there was only one error, and the decoder produced the correct codeword. See also Example 2.18.

This Hamming code can transmit 4 information source bits for seven channel uses, so it is more efficient than the repeat code. But, the repeat code is more powerful, because it can correct 3 errors, while the Hamming code can only correct one error. This is a fundamental tradeoff in coding theory: the efficiency of the code (later defined as the code's rate) versus its error correction capability (later defined as the code's minimum distance).

SSQ 1.1. For the (7,4) Hamming code in Fig. 1.2, find the estimated codeword $\widehat{\mathbf{c}}$ for each $\mathbf{y}$:

- $\mathbf{y} = [0, 0, 0, 0, 0, 0, 1]$,
- $\mathbf{y} = [0, 0, 0, 1, 0, 0, 0]$

Go to the course website for solutions and more SSQs.

1.2 Communications System Model

A formal description of the system in Fig. 1.1 is given. An *information source* produces one of M symbols $u \in \{1, 2, \ldots, M\}$. Each of the M symbols is equally likely.

Definition 1.1. An *error-correcting code* $\mathcal{C}$ consists of M codewords:

$$\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_M\} \tag{1.8}$$

where each codeword (or constellation point) is an n-tuple:

$$\mathbf{c} = [c_1, c_2, \ldots, c_n]. \tag{1.9}$$

Each c_i is from a specified alphabet, often the binary alphabet.

Definition 1.2. The *code rate* R of a code is:

$$R = \frac{1}{n} \log_2 M \tag{1.10}$$

Example 1.1. For the repeat code in Subsection 1.1.2, $M = 2$ and $n = 7$, so the rate is $R = \frac{1}{7}$.

For the Hamming code in Subsection 1.1.3, $M = 16$ and $n = 7$, so the rate is $R = \frac{4}{7}$.

The code rate represents the number of information bits per codeword symbol. Consider the case that M is a power of 2, say $M = 2^k$. Then the message can be described using k bits, and the code rate is $R = k/n$, the number of bits of information, divided by the number of times the channel was used. The higher the rate, the more information the code can carry.

An *encoder* is a mapping from information symbols to codewords. If the information alphabet is $\mathcal{U}$, then:

$$\text{Encode} : \mathcal{U} \to \mathcal{C}.$$

The information source symbol u is one-to-one encoded to its codeword $\mathbf{c}_u$. For the repeat code $\mathcal{U} = \{0, 1\}$ and for the Hamming code $\mathcal{U} = \{0000, 0001, \ldots, 1111\}$.

The codeword is transmitted over a noisy *channel*. The output of the channel is a sequence $\mathbf{y}$ of n symbols:

$$\mathbf{y} = [y_1, y_2, \ldots, y_n]. \tag{1.11}$$

The channel output may be continuous, $y_i \in \mathbb{R}$, or discrete, y_i from a finite set. A probabilistic model of the channel is used, meaning that $\Pr[y_i|c_i]$ is given, and examples are given in Section 1.3.

The *decoder* receives this sequence, and then outputs an estimate of the information source $\widehat{\mathbf{u}}$, or an estimate of the transmitted codeword $\widehat{\mathbf{c}} = [\widehat{c}_1, \widehat{c}_2, \ldots, \widehat{c}_n]$. If the channel output alphabet is $\mathcal{Y}$, then the decoder is the mapping:

$$\text{Decode} : \mathcal{Y}^n \to \mathcal{C}. \tag{1.12}$$

Since the encoding is one-to-one, if we know $\widehat{\mathbf{c}}$ then the corresponding $\widehat{\mathbf{u}}$ is easily found. The decoder knows the codebook $\mathcal{C}$, and generally finds the codeword $\widehat{\mathbf{c}} \in \mathcal{C}$ which is "closest" in some sense, to $\mathbf{y}$.

The central objective is that the decoder output $\widehat{\mathbf{u}}$ should be the same as the encoder input $\mathbf{u}$, with high probability. If $\mathbf{u} = \widehat{\mathbf{u}}$, then the decoded succeeded with no errors. If $\mathbf{u} \neq \widehat{\mathbf{u}}$, then a decoding error occurred.

1.3 Channel Models

The channel shown in Fig. 1.1 introduces noise into the communication system. Two channel models are considered. The discrete memoryless channel (DMC) has discrete outputs. The additive white Gaussian noise (AWGN) channel has continuous-valued outputs.

The general channel has input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$ and output probability $p_{\mathsf{Y}|\mathsf{X}}(y|x)$. In the vector channel model is n uses of the channel, so the channel input sequence is $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and the channel output sequence is $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The channel is described by joint conditional distribution $p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$. Because the channel is memoryless, we have:

$$\begin{aligned} p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) &= p_{\mathsf{Y}|\mathsf{X}}(y_1|x_1)p_{\mathsf{Y}|\mathsf{X}}(y_2|x_2)\cdots p_{\mathsf{Y}|\mathsf{X}}(y_n|x_n) \\ &= \prod_{i=1}^{n} p_{\mathsf{Y}|\mathsf{X}}(y_i|x_i). \end{aligned}$$

Note that $\mathcal{X}$ is both the codebook alphabet and the channel input alphabet.

1.3.1 Discrete Memoryless Channel

Definition 1.3. A *discrete memoryless channel* (DMC) consists of an input alphabet $\mathcal{X}$, an output alphabet $\mathcal{Y}$ and a conditional probability distribution $p_{\mathsf{Y}|\mathsf{X}}(y|x)$.

The codeword has n symbols $\mathbf{c} = [c_1, c_2, \ldots, c_n]$. We use c_i to denote a codeword symbol and x_i to denote the channel input. For DMCs, these two are equal:

$$c_i = x_i, \tag{1.13}$$

for $i = 1, \ldots, n$, so the channel input is $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. The output of the channel is the n symbols $\mathbf{y} = [y_1, y_2, \ldots, y_n]$. To transmit n codeword symbols, we use the channel n times. The DMC is memoryless and the conditional distribution also factors as (1.21). Two important examples are the binary symmetric channel and the binary erasure channel.

Binary Symmetric Channel The binary symmetric channel (BSC) is a discrete memoryless channel where an error occurs with probability α. It has binary inputs $\mathcal{X} = \{0, 1\}$, binary outputs $\mathcal{Y} = \{0, 1\}$. For a parameter α, the probability transition matrix $p_{\mathsf{Y}|\mathsf{X}}(y|x)$ is:

$$p_{\mathsf{Y}|\mathsf{Z}}(y|z) = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}. \tag{1.14}$$

That is, the channel makes an error with probability p, where $0 \leq p \leq 1$. For the BSC, an error occurs if an input 0 is changed to a 1, and this happens with probability p. Similarly, an input 1 changes to a 0 with probability p. There is no error with probability $1 - \alpha$.

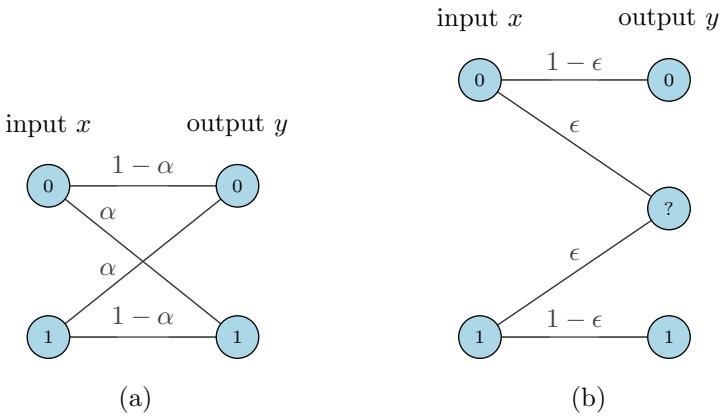


Figure 1.3: (a) Binary symmetric channel (BSC) with error probability α. (b) Binary erasure channel with erasure probability ϵ.

Example 1.2. Consider a BSC with error probability $p = 0.2$. The channel transition probabilities $p_{\mathsf{Y}|\mathsf{X}}(y|x)$ are:

$$p_{\mathsf{Y}|\mathsf{X}}(y|x) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}. \tag{1.15}$$

If the transmitter sends $x = 0$, then $y = 0$ is correctly received with probability 0.8. An error occurs if $y = 1$ is received; this occurs with probability 0.2. The probabilities on y are reversed if $x = 1$ is transmitted. The following is an example of transmitted $\mathbf{x}$ and received $\mathbf{y}$:

$\mathbf{x} =$	1	1	1	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1
$\mathbf{y} =$	1	1	1	1	0	1	1	1	0	0	1	0	1	0	1	1	0	0	0	1
					E	E		E						E	E					

The errors are marked by "E."

Binary Erasure Channel The binary erasure channel (BEC) is a discrete memoryless channel also has binary inputs $\mathcal{Y} = \{0, 1\}$. The outputs are $\mathcal{Y} = \{0, ?, 1\}$ where ? is an erasure symbol. For a parameter ϵ, the probability transition matrix $p_{\mathsf{Y}|\mathsf{X}}(y|x)$ is:

$$p_{\mathsf{Y}|\mathsf{X}}(y|x) = \begin{bmatrix} 1-\epsilon & \epsilon & 0 \\ 0 & \epsilon & 1-\epsilon \end{bmatrix}, \tag{1.16}$$

where $0 \leq \epsilon \leq 1$. For the BEC, an input symbol 0 or 1 is erased with probability ϵ, and received correctly with probability $1 - \epsilon$. It is not possible for an input 0 to become an output 1 (or 1 to become a 0).

1.3.2 Additive White Gaussian Noise (AWGN) Channel

The *Gaussian distribution* (or normal distribution) $p_{\mathsf{Z}}(z)$ with mean m, variance σ^2 is:

$$p_{\mathsf{Z}}(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-m)^2}{2\sigma^2}} \text{ for } z \in \mathbb{R}. \tag{1.17}$$

We often abbreviate the Gaussian distribution as $\mathsf{Z} \sim \mathcal{N}(m, \sigma^2)$.

The AWGN channel model is a common model of communications which has Gaussian noise.

Definition 1.4. The *additive white Gaussian noise* (AWGN) channel model with power constraint P and noise power σ^2 has n inputs $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and n outputs $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ where:

$$y_i = x_i + z_i, \tag{1.18}$$

the $z_i \sim \mathcal{N}(0, \sigma^2)$ are independent and identically distributed Gaussian distributed with mean 0 and variance σ^2, for $i = 1, 2, \ldots, n$. Furthermore, the x_i satsify:

$$\frac{1}{n} \sum_{i=1}^{n} x_i^2 \leq P. \tag{1.19}$$

For each $x \in \mathcal{X}$, the probability that symbol x is transmitted is $p_{\mathsf{X}}(x)$. The *average transmit power* E_s is given by:

$$E_s = \sum_{x \in \mathcal{X}} p_{\mathsf{X}}(x) x^2. \tag{1.20}$$

The AWGN channel is memoryless, which means the errors at any time i are independent of errors at any other time. That is, the joint distribution probability on the channel output given the channel input is $p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$. Because the channel is memoryless, this factors as:

$$p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p_{\mathsf{Y}|\mathsf{X}}(y_i|x_i) \tag{1.21}$$

Due to the Gaussian noise distribution, $p_{\mathsf{Y}|\mathsf{X}}(y|x)$ is given by:

$$p_{\mathsf{Y}|\mathsf{X}}(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-x)^2/2\sigma^2} \tag{1.22}$$

An example with $n = 1$ is the *binary-input additive white Gaussian noise* (BI-AWGN) channel or the *binary phase-shift keying* (BPSK) channel. This channel has input $\mathcal{X} = \{+1, -1\}$, and $p_{\mathsf{X}}(x) = [\frac{1}{2}, \frac{1}{2}]$. The conditional channel output distribution is:

$$p_{\mathsf{Y}|\mathsf{X}}(y|-1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y+1)^2/2\sigma^2} \tag{1.23}$$

$$p_{\mathsf{Y}|\mathsf{X}}(y|+1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-1)^2/2\sigma^2} \tag{1.24}$$

The average transmit power is:

$$E_s = \frac{1}{2}(-1)^2 + \frac{1}{2}(+1)^2 = 1 \tag{1.25}$$

1.4 Channel Capacity

Now consider using an error-correcting code to communicate over a noisy channel. The error correcting code consists of n symbols $c_1, c_2, \ldots, c_n$ (or $x_1, x_2, \ldots, x_n$), which are input to the channel. The output of the channel are n symbols $y_1, y_2, \ldots, y_n$. The decoder attempts to recover the transmitted data, and produces either an estimate of the codeword $\widehat{\mathbf{c}}$ or an estimate of the information $\widehat{\mathbf{u}}$.

The performance of the decoder is important in evaluating communication systems using error-correcting codes. Two common metrics are the word error rate (WER) and the bit error rate (BER). A word error occurs if $\mathbf{u} \neq \widehat{\mathbf{u}}$, or equivalently if $\mathbf{c} \neq \widehat{\mathbf{c}}$. Then, the word-error rate is:

$$\text{WER} = \Pr(\mathbf{u} \neq \widehat{\mathbf{u}} | \mathbf{u} \text{ was transmitted}, \mathbf{y} \text{ was received}). \tag{1.26}$$

Likewise, a bit error occurs if $u_i \neq \widehat{u}_i$, so the BER is:

$$\text{BER} = \Pr(u_i \neq \widehat{u}_i | u_i \text{ was transmitted}, \mathbf{y} \text{ was received}) \tag{1.27}$$

Finding the WER and BER analytically is impossible is most cases of interest, and Monte Carlo methods are commonly used to estimate WER and BER; see Chapter 3.

1.4.1 Channel Coding Theorem

The channel capacity is the maximum possible communications rate for a channel. Generally speaking, error-correcting codes improve as the block length n gets larger. When discussing channel capacity, reliable communications means we can make the WER go to 0 as $n \to \infty$.

Let X and Y be jointly distributed random variables. The *mutual information* between X and Y is denoted $I(\mathsf{X};\mathsf{Y})$.

Definition 1.5. Consider random variables X and Y with a joint probability distribution function $p_{\mathsf{X},\mathsf{Y}}(x,y)$ and marginal distributions $p_{\mathsf{X}}(x)$ and $p_Y(y)$. Then $I(\mathsf{X};\mathsf{Y})$ is given by:

$$I(\mathsf{X};\mathsf{Y}) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{\mathsf{X},\mathsf{Y}}(x,y) \log \frac{p_{\mathsf{X},\mathsf{Y}}(x,y)}{p_{\mathsf{X}}(x) p_{\mathsf{Y}}(y)}. \tag{1.28}$$

While a channel $p_{\mathsf{Y}|\mathsf{X}}(y|x)$ is fixed, the input distribution $p_{\mathsf{X}}(x)$ is not. The channel capacity is the maximum value of $I(\mathsf{X};\mathsf{Y})$ over all $p_{\mathsf{X}}(x)$.

Definition 1.6. For a discrete memoryless channel $p_{\mathsf{Y}|\mathsf{X}}(y|x)$, the *channel capacity* C of a memoryless channel is:

$$C = \max_{p_{\mathsf{X}}(x)} I(\mathsf{X};\mathsf{Y}). \tag{1.29}$$

In addition, an optimal $p^*_{\mathsf{X}}(x)$ is called the capacity-achieving input distribution:

$$p^*_{\mathsf{X}}(x) = \arg\max_{p_{\mathsf{X}}(x)} I(\mathsf{X};\mathsf{Y}). \tag{1.30}$$

An important result in information theory states that for reliable communication over a channel, than the code rate R must be less than the capacity C.

Proposition 1.1. *Channel Coding Theorem* For every rate $R < C$, there exists a sequence of $(2^{nR}, n)$ codes with probability of decoding error going to 0 as $n \to \infty$. Conversely, any sequence of $(2^{nR}, n)$ codes with probability of decoding error going to 0 must have $R \leq C$.

That is, there exists a code with rates $R < C$ for which reliable communications is possible. On the other hand, for any code with $R > C$, reliable communication is not possible.

1.4.2 Capacity of BSC and BEC

Define the binary entropy function as $h(p) = -p\log p - (1-p)\log(1-p)$.

Proposition 1.2. The capacity of the binary symmetric channel (BSC) with error probability α is:

$$C = 1 - h(\alpha)$$

with capacity-achieving input distribution $p^*_{\mathsf{X}}(x) = [\frac{1}{2}, \frac{1}{2}]$.

Proposition 1.3. The capacity of the binary erasure channel (BEC) with erasure probability p is:

$$C = 1 - p$$

with capacity-achieving input distribution $p^*_{\mathsf{X}}(x) = [\frac{1}{2}, \frac{1}{2}]$.

1.4.3 Capacity of the AWGN Channel

For the AWGN channel, the capacity has a closed form solution. This is one of the most elegant results from information theory.

Proposition 1.4. The capacity of AWGN channel with power constraint P and noise variance σ^2 is:

$$C = \frac{1}{2}\log\left(1 + \frac{P}{\sigma^2}\right) \text{ bits per transmission} \tag{1.31}$$

The capacity-achieving input distribution is $p^*_{\mathsf{X}}(x)$ is a zero-mean Gaussian with variance P.

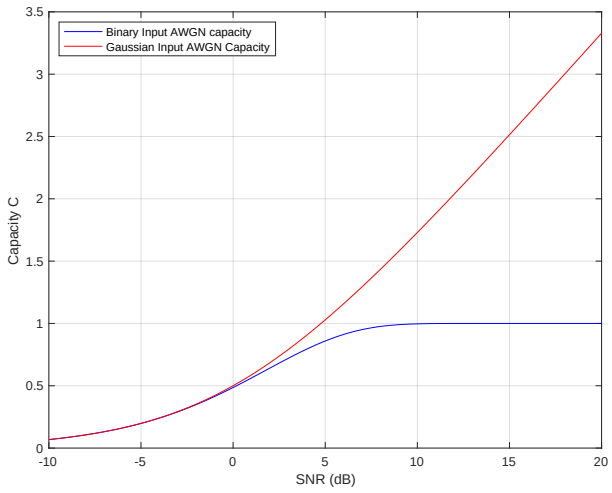


Figure 1.4: Capacity of the AWGN channel, with Gaussian inputs and binary inputs (BI-AWGN).

1.4.4 Capacity of the BI-AWGN Channel

Capacity can be achieved with a Gaussian input distribution. It is convenient to consider the special case when the input alphabet is limited to $\mathcal{X} = \{-1, +1\}$. This is the binary-input AWGN (BI-AWGN) channel. This is particularly convenient using a binary code, using the mapping from the binary code $\{0, 1\}$ to the channel input $\{-1, +1\}$. The capacity of this channel is given by:

$$C = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} e^{-(y-1)^2/2\sigma^2} \log_2 \frac{2}{1 + e^{-2y/\sigma^2}} dy \tag{1.32}$$

and is achieved by choosing $p_{\mathsf{X}}(0) = p_{\mathsf{X}}(1) = \frac{1}{2}$. Unfortunately, this important channel does not have a closed-form solution for its capacity. To achieve the capacity, we need to use a code with $n \to \infty$. On the other hand, the case of $n = 1$ was considered in Subsection 1.3.2.

The capacity of the AWGN channel and the BI-AWGN channel is plotted in 1.4. For channels with low SNR, the capacity is the same. However, for channels with high SNR, the capacity of the BI-AWGN is limited to 1, since the inputs are binary. On the other hand, the capacity of the AWGN channel is unbounded.

1.4.5 The Challenge of Coding Theory

Now we come to challenge of coding theory — how do we design codes for reliable communications?

While the channel coding theorem perspective is "given a channel, what is the highest possible rate?". However, it is equally valid to ask "give a rate, what is worst channel we can use". The case of $R = \frac{1}{2}$ is considered in Fig. 1.5

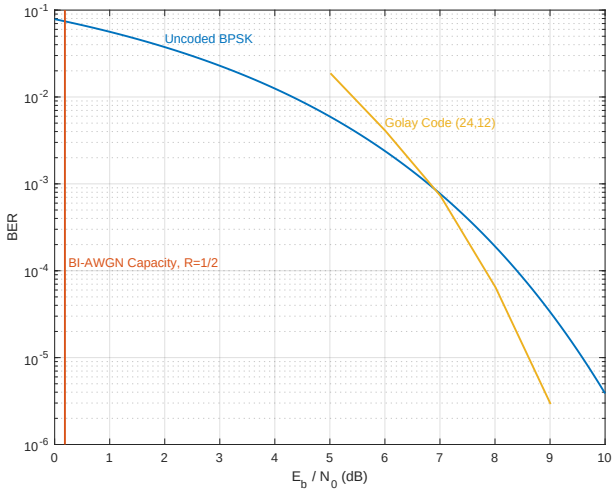


Figure 1.5: Bit-error rate for the uncoded BPSK, the extended Golay code (24,12), and the capacity of the AWGN channel

For the AWGN channel, the signal-to-noise ratio (SNR) is a measure of how noisy the channel is. The definition of the signal-to-noise ratio is quite important, and here E_b/N_0 will be used. The power per information symbol is $E_b = E_s/R$. The noise power is $N_0 = 2\sigma^2$ where σ^2 the the variance of the Gaussian noise. Thus the SNR definition E_b/N_0 is:

$$\frac{E_b}{N_0} = \frac{E_s}{2\sigma^2 R} \tag{1.33}$$

where E_s is given in (1.20). Also, E_b/N_0 is often expressed in dB, that is:

$$E_b/N_0(dB) = 10\log_{10} E_b/N_0. \tag{1.34}$$

Consider first uncoded communication on the BI-AWGN channel, we can find the probability of error. The decoder receives y, and uses the following *hard decision* rule to estimate the transmitted information:

$$\widehat{x} = \begin{cases} -1 & \text{if } y < 0 \\ +1 & \text{if } y \geq 0 \end{cases}. \tag{1.35}$$

An error occurs if $\widehat{x} \neq x$. The exact probability of error has an analytical expression:

$$\begin{aligned} \Pr(x \neq \widehat{x}) &= \frac{1}{2}\Pr(y > 0 | x = -1) + \frac{1}{2}\Pr(y < 0 | x = 1) && (1.36)\\ &= \Pr(y > 0 | x = -1) && (1.37)\\ &= \int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+1)^2}{2\sigma^2}} dy && (1.38)\\ &= \frac{1}{2}\text{erfc}(\frac{1}{\sqrt{2\sigma^2}}), && (1.39) \end{aligned}$$

where erfc is the complementary error function.

As can be seen in the figure, there is a large gap between capacity and the uncoded scheme, for any BER. Additionally has been shown the $n = 24$ Golay code, which is an excellent code for this block length. However, in order to get closer to capacity, we need longer and more powerful codes. This is the challenge of coding theory.

1.5 Groups, Rings and Fields

Group, rings and fields are algebraic structures that are central to understanding error-correcting codes. They are studied extensively under the heading of "abstract algebra" "group theory" or "field theory". This section has an informal treatment, with more details in Chapter **??**. Wikipedia: Abstract algebra

Roughly speaking, a *group* is set where addition and subtraction are defined. A *ring* is a set where addition, subtraction and multiplication are defined. A *field* is a set where addition, subtraction, multiplication and division are defined. Any field is a ring. Any ring is a group. "Addition" and "multiplication" may not be familiar the operations on the numbers, but must be defined with respect to the group, ring or field.

The integer multiples of five, $5\mathbb{Z}$ is an example of a group with respect to usual addition:

$$5\mathbb{Z} = \{\ldots, -10, -5, 0, 5, 10, \ldots, \}. \tag{1.40}$$

The sum of any two numbers which are a multiple of five, is itself a multiple of five: $10 + 15 = 25$ is a multiple of five. Subtraction is also defined, for example $15 - 25 = -10$. The additive identity is 0, that is $5 + 0 = 5$. (But the multiplicative identity 1 is not in this set, so it is not a ring — see Chatper **??**.)

An example of a ring $\mathcal{R}$ is the integers:

$$\mathcal{R} = \{0, 1, 2, 3, 4, 5\} \tag{1.41}$$

with addition and multiplication taken modulo 6. For example, $3+5 \bmod 6 = 2$ is addition in the ring and $2 \cdot 5 \bmod 6 = 4$ is multiplication within the ring. This ring is not a field, because division is not well defined, for example 2^{-1} does not exist, because there is no element a such that $2 \cdot a = 1$. Another example of a ring are the integers $\mathbb{Z}$ — the sum of any two integers is an integer and the product of any two integers is an integer.

The set of real numbers $\mathbb{R}$ is an example of a field, since addition, subtraction, multiplication and division are well defined. However, we are particularly interested in finite fields[1] $\mathbb{F}_q$ of size q, which are fields defined on a finite set of size q. The binary field $\mathbb{F}_2$ has addition and multiplication defined on $\{0, 1\}$:

$+$	0	1
0	0	1
1	1	0

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

[1] The finite field $\mathbb{F}_q$ is also called a Galois field, denoted GF(q)

Binary arithmetic is the same as conventional arithmetic, except $1 + 1 = 0$. Note that $-1 = 1$ and $-0 = 0$, so for any a, $a = -a$.

The ternary field $\mathbb{F}_3$ on $\{0, 1, 2\}$ has addition and multiplication defined with respect to usual operations modulo 3:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

The additive inverse of a is in the column where row a has a 0. For example,$-2 = 1$, that is $2 + (-2) = 2 + 1 = 0$. Similarly, the multiplicative inverse of a is the column where row a has a 1. For example, $2^{-1} = 2$ since $2 \cdot 2 = 1$ means $2 = 2^{-1}$. If q is a prime number, a field of size q is formed using addition and multiplication modulo q.

Fields exist for some sizes which are not a prime number. Consider the field $\mathbb{F}_4$ of size 4 on the set $\{0, 1, \alpha, \alpha^2\}$. Here α and α^2 are two distinct elements of the field. The addition and multiplication operations are given by:

$+$	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

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

As usual, 0 is the additive identity: $a + 0 = a$, and 1 is the multiplicative identity: $1 \cdot a = a$. Note that addition and multiplication modulo 4 does not give a finite field. Subtraction and division can be found from the tables above. For example, $2 + (-2) = 0$ is an example of subtraction. The multiplicative inverse exists because there is a 1 in each non-zero row and column of the multiplication table, for example $2^{-1} = 3$ because $2 \cdot 3 = 1$.

Fields exist only for sizes that are prime numbers, or a power of a prime number. Since $4 = 2^2$ is a power of a prime number, a field of size 4 exists, as was shown above. But there exist no fields of size 6 or 10, for example, since these numbers cannot be written as the power of a prime number. This is discussed in Chapter **??**.

Familiar algebra can be performed using fields. For example, let $f(x) = x^2 - x + 1$ be a polynomial where x and coefficients are from $\mathbb{F}_3$. The polynomial can be evaluated in the field: $f(0) = 0 - 0 + 1 = 1$, $f(1) = 1 - 1 + 1 = 1$ and $f(2) = 2^2 - 2 + 1 = 0$. As an example in $\mathbb{F}_4$, let $f(x) = x^2 + \alpha^2 x + \alpha$. Using the $\mathbb{F}_4$ addition and multiplication tables: $f(0) = \alpha$, $f(1) = 1^2 + \alpha^2 + \alpha = 0$, $f(\alpha) = \alpha^2 + \alpha^2 \cdot \alpha + \alpha = \alpha^2 + 1 + \alpha = 0$ and $f(\alpha^2) = (\alpha^2)^2 + (\alpha^2) \cdot (\alpha^2) + \alpha = \alpha + \alpha + \alpha = \alpha$.

Linear systems of equations can be formed using finite fields. For example, let x, y, z be elements in $\mathbb{F}_2$. The following set of equations can be solved to find

the unknowns:

$$x + y \quad = 1 \tag{1.42}$$
$$x \quad + z = 1 \tag{1.43}$$
$$x + y + z = 0 \tag{1.44}$$

It can be solved by subtracting the first equation from the third equation to find $z = 1$, since $-1 = 1$. Using $z = 1$ in the second equation, $x = 0$. Finally using the first equation, $y = 1$.

SSQ 1.2. *Finite fields.* Using the table of addition and subtraction for $\mathbb{F}_4$, find (a) -1 (b) α^{-1} (c) if $f(x) = x^2 + 1$, then find $f(\alpha^2)$.

SSQ 1.3. *Algebraic Structures.* Using the table of addition and subtraction for $\mathbb{F}_4$, find (a) -1 (b) 2^{-1} (c) if $f(x) = x^2 + 1$, then find $f(3)$.

1.6 Exercises

1.1 Describe how to cut 88 circles of 1-inch diameter out of a sheet of paper of width 8.5 inches and length 11 inches. Prove that it is not possible to cut on more than 119 circles of 1-inch diameter.

1.2 In $\mathbb{R}^n$, a hypercube of side length ℓ has volume ℓ^n. Give an upper bound on the number of n-balls of radius $r = 1$ that fit inside this hypercube.

1.3 Prove that it is not possible to find 32 binary codewords, each of length 8 bits, such that every word differs from every other word in at least three places.

1.4 For any block code with length n over alphabet size q with minimum distance $2t + 1$ or greater, the number of codewords M satisfies:

$$M \leq \frac{q^n}{\left(1 + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right)}. \tag{1.52}$$

Prove this statement.

1.5 Consider the binary-input AWGN system, with input $x \in \{-1, +1\}$ and Gaussian noise with variance σ^2. The decoder uses the hard decision rule (1.11) to make the estimate $\widehat{x}$.

(a) Using your favorite programming language, write a simulation to estimate the probability of error for this system (an error occurs if $\widehat{c} \neq c$). Transmit the symbol N times. Count the number of times $\widehat{c} \neq c$, call this N_{error}. Using N sufficiently large, use your simulation to estimate the probability of error:

$$\frac{N_{error}}{N} \tag{1.56}$$

for $\sigma^2 = 1, 0.5, 0.15$.

(b) The exact probability of error has an analytical expression given in (1.26). Compare the analytic value (some languages have a built-in erfc function) with the simulation in part (a).

1.6 Solve the following algebraic problems over $\mathbb{F}_2$ $\mathbb{F}_3$ or $\mathbb{F}_4$, as indicated. Use the tables in Section 1.2.

(a) Let $f(x) = x^4 + x^3 + x + 1$ be a polynomial over $\mathbb{F}_2$. Evaluate $f(0)$, $f(1)$

(b) Let $f(x) = x^2 - 1$ be a polynomial over $\mathbb{F}_3$. Evaluate: $f(0)$, $f(1)$, $f(2)$

(c) Let $f(x) = x^3 - 1$ be a polynomial over $\mathbb{F}_4$. Evaluate: $f(0)$, $f(1)$, $f(2)$, $f(3)$,

(d) Solve the following system over $\mathbb{F}_2$ for a, b, c

$$\begin{aligned} a + b + c &= 1 \\ b - c &= 0 \\ a + b &= 1 \end{aligned}$$

(e) Solve the following system over $\mathbb{F}_3$ for a, b, c

$$\begin{aligned} a + 2b + c &= 2 \\ b - 2c &= 1 \\ 2a + b &= 1 \end{aligned}$$

(f) Compute the determinant and the inverse of the following matrix over $\mathbb{F}_4$:

$$\begin{bmatrix} 3 & 3 \\ 2 & 1 \end{bmatrix}$$

1.7 The *extended* Hamming code has a graphical representation created by adding a new parity bit p_4 outside of the three circles of the original Hamming code representation of Fig. 1.2. Add a fourth circle around all 8 bits. The new codeword is:

$$[u_1, u_2, u_3, u_4, p_1, p_2, p_3, p_4]$$

As before, the number of 1's inside each circle, including the new fourth circle, must be even.

(a) Draw a graphical representation of the extended Hamming code.

(b) Encode $[u_1, u_2, u_3, u_4] = [0, 1, 1, 1]$ to its codeword.

(c) Decode $\mathbf{y} = [0, 0, 1, 1, 0, 0, 1, 0]$ and $\mathbf{y} = [0, 1, 1, 0, 0, 1, 1, 1]$ to the Hamming codeword. The extended Hamming code can correct one error.

(d) A codeword was transmitted over the erasure channel and $\mathbf{y} = [?, 1, 0, ?, 0, 1, ?, 0]$ was received. Decode $\mathbf{y}$ to $\widehat{\mathbf{c}}$.

.

Chapter 2

Linear Block Codes

Almost all error-correction codes used in practice are linear codes. This chapter introduces linear block codes.

2.1 Finite-Field Codes

2.1.1 Definition

This section describes finite-field codes, which are error-correcting codes defined using n symbols from a finite field. These are commonly called block codes. A block code is a set of vectors in $\mathbb{F}_q^n$.

Definition 2.1. A *finite-field code* or *block code* $\mathcal{C}$ with block length n is the set of M codewords

$$\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_M\} \tag{2.1}$$

and each codeword c has n symbols:

$$\mathbf{c} = (c_1, c_2, \ldots, c_n). \tag{2.2}$$

where each $c_i \in \mathbb{F}_q$.

We say n is the block length of the code. We say the code is defined over the finite field $\mathbb{F}_q$.

Definition 2.2. The rate R of a block length n code with M codewords is:

$$R = \frac{1}{n} \log M \tag{2.3}$$

Example 2.1. The following code has $n = 4, q = 5$ and $M = 3$:

$$\begin{aligned} \mathcal{C} = \{&(0,0,0,0),\\ &(3,0,4,1),\\ &(2,2,2,3)\} \end{aligned}$$

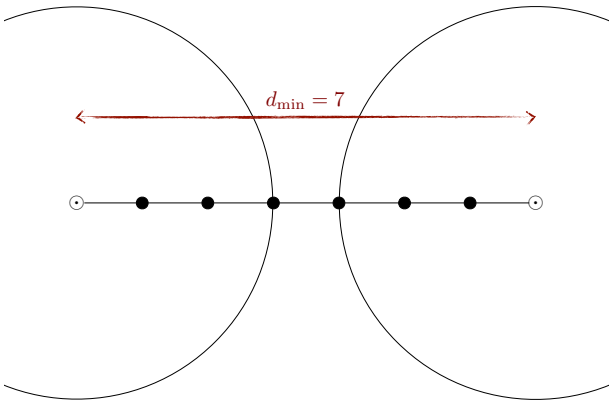


Figure 2.1: Minimum distance between two codewords (denoted $\odot$) is 7. The correctable number of errors is 3.

The minimum distance is an important property of a code $\mathcal{C}$. If two codewords $\mathbf{x}$ and $\mathbf{y}$ are close in minimum distance, if $\mathbf{x}$ is transmitted, then noise might cause $\mathbf{y}$ to be decoded erroneously.

Definition 2.3. The *Hamming weight* $\mathrm{wt_H}(\mathbf{x})$ of a vector $\mathbf{x}$ is number of nonzero elements in $\mathbf{x}$. Mathematically that is written as:

$$\mathrm{wt_H}(\mathbf{x}) = |\{i|x_i \neq 0\}| \tag{2.4}$$

Definition 2.4. The *Hamming distance* $d_{\mathrm{h}}(\mathbf{x}, \mathbf{y})$ or $d(\mathbf{x}, \mathbf{y})$ between two vectors $\mathbf{x}$ and $\mathbf{y}$ is the number of positions where $\mathbf{x}$ and $\mathbf{y}$ differ:

$$d(\mathbf{x}, \mathbf{y}) = |\{i|x_i \neq y_i\}| \tag{2.5}$$

Example 2.2. The Hamming weight of the binary vector [1 0 1 1 1 1 0] is 5. The Hamming weight of the non-binary vector [2 0 3 0 0 4] is 3.

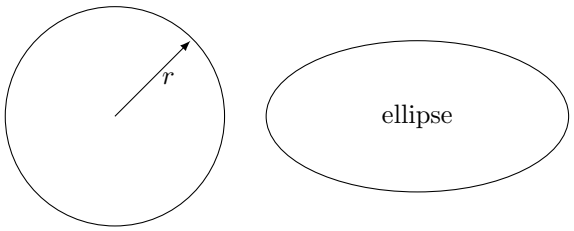
Example 2.3. The Hamming distance between the binary vectors:

$$[1\ 0\ 1\ 0\ 0] \text{ and} \tag{2.6}$$
$$[1\ 1\ 0\ 0\ 1] \tag{2.7}$$

is 3. The Hamming distance between the non-binary vectors:

$$[0\ 1\ 5\ 4\ 0] \text{ and} \tag{2.8}$$
$$[0\ 2\ 5\ 1\ 0] \tag{2.9}$$

is 2.



SSQ 2.1. *Hamming Distance* What is the Hamming weight of $(1, 2, 3, 0, 0)$? What is the Hamming distance between $(0, 0, 0, 1, 1, 0)$ and $(1, 0, 1, 1, 0, 0)$?

Some properties of Hamming weight and Hamming distance:

$$d_{\mathrm{h}}(\mathbf{x}, \mathbf{0}) = \mathrm{wt}_{\mathrm{H}}(\mathbf{x}) \tag{2.10}$$

$$d_{\mathrm{h}}(\mathbf{x}, \mathbf{y}) = \mathrm{wt}_{\mathrm{H}}(\mathbf{x} - \mathbf{y}) = \mathrm{wt}_{\mathrm{H}}(\mathbf{y} - \mathbf{x}) \tag{2.11}$$

$$d_{\mathrm{h}}(\mathbf{x}, \mathbf{y}) \leq d_{\mathrm{h}}(\mathbf{x}, \mathbf{w}) + d_{\mathrm{h}}(\mathbf{w}, \mathbf{y}) \qquad \text{the triangle inequality} \tag{2.12}$$

The concept of a Hamming ball and its volume is introduced. "Volume" means number of sequences. In the vector space $\mathbb{F}_q^n$ there are q^n sequences, so we say the volume of $\mathbb{F}_q^n$ is q^n.

Definition 2.5. A *Hamming ball* of radius t around a point $\mathbf{c}$ is the set of sequences with Hamming distance t or less from $\mathbf{c}$:

$$\{\mathbf{y} | d(\mathbf{c}, \mathbf{y}) \leq t\}. \tag{2.13}$$

The number of sequences in the Hamming ball of radius t is its volume $V_n(t)$:

$$V_n(t) = \sum_{i=0}^{t} (q-1)^i \binom{n}{i} \tag{2.14}$$

where $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

Example 2.4. With $q = 2$ and $n = 4$, the Hamming ball of radius $t = 1$ around the point $(0, 0, 0, 0)$ is:

$$\{(0,0,0,0), (0,0,0,1), (0,0,1,0), (0,1,0,0), (1,0,0,0)\}. \tag{2.15}$$

The volume here is $V_4(1) = 1 + \binom{4}{1} = 5$, because there are 5 sequences. If the radius is increased to $t = 2$, then the Hamming sphere additionally includes:

$$\{(0,0,1,1), (0,1,0,1), (1,0,0,1), (0,1,1,0), (1,0,1,0), (1,1,0,0)\}. \tag{2.16}$$

and the volume is $V_4(2) = 1 + \binom{4}{1} + \binom{4}{2} = 11$.

2.1.2 Minimum Distance and Error Correction

Recall the DMC model of a communications channel. The errors due to the channel is modeled using an error vector $\mathbf{e}$:

$$\mathbf{e} = (e_1, e_2, \ldots, e_n) \tag{2.17}$$

Then the received sequence $\mathbf{y}$ is:

$$\mathbf{y} = \mathbf{c} + \mathbf{e}, \tag{2.18}$$

where addition is in the field of the code. The weight of the error vector $\mathrm{wt_H}(\mathbf{e})$ is equal to the number of errors. The error vector from Example 1.1.2 is $\mathbf{e} = [1\ 0\ 0\ 1\ 0\ 1\ 0]$, a three-errors vector; the weight of $\mathbf{e}$ is $\mathrm{wt_H}(\mathbf{e}) = 3$.

Definition 2.6. The *minimum distance* $d_{\min}$ of a block code is the minimum of the Hamming distances between all distinct pairs of codewords $\mathbf{x}$ and $\mathbf{y} \in \mathcal{C}$:

$$d_{\min} = \min_{\mathbf{x},\mathbf{y}\in\mathcal{C},\mathbf{x}\neq\mathbf{y}} d(\mathbf{x}, \mathbf{y}) \tag{2.19}$$

Example 2.5. Consider a code with $M = 5$ codewords:

$$\begin{aligned}
\mathbf{c}_1 &= (1, 0, 1, 1, 1, 0) \\
\mathbf{c}_2 &= (1, 0, 0, 1, 0, 1) \\
\mathbf{c}_3 &= (1, 1, 1, 0, 0, 1) \\
\mathbf{c}_4 &= (0, 1, 0, 1, 1, 1) \\
\mathbf{c}_5 &= (0, 0, 0, 0, 0, 0)
\end{aligned}$$

The distance between all pairs is given by:

$d(\cdot,\cdot)$	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$
$\mathbf{c}_1$	0	3	4	4	4
$\mathbf{c}_2$	3	0	3	3	3
$\mathbf{c}_3$	4	3	0	4	4
$\mathbf{c}_4$	4	3	4	0	4
$\mathbf{c}_5$	4	3	4	4	0

The minimum distance between distinct codewords is 3, so for this code, $d_{\min} = 3$.

Principle of Hamming distance decoding Given a received sequence $\mathbf{y}$, a decoder should choose as its output a codeword $\widehat{\mathbf{c}}$ which is closest in Hamming distance to $\mathbf{y}$.

Example 2.6. Continuing Example 2.5, assume that

$$\mathbf{y} = [0, 0, 0, 1, 1, 1] \tag{2.20}$$

is received. The Hamming distance to each codeword is:

	$d(\mathbf{c}_i, \mathbf{y})$
$\mathbf{c}_1$	3
$\mathbf{c}_2$	2
$\mathbf{c}_3$	5
$\mathbf{c}_4$	1
$\mathbf{c}_5$	3

The closest codeword in the Hamming distance sense is $\mathbf{c}_4$, so the decoder outputs $\widehat{\mathbf{c}} = \mathbf{c}_4$. Note that ties are possible — to resolve the ties, the the decoder may randomly choose one of the minimum distance codewords randomly.

If the number of errors is not too large, then it may be possible to correct the errors, and correctly decode the transmitted codeword. The Hamming distance is important for determining the error-correction capability of the code. *A t-error correcting code* can correctly determine the transmitted codeword if the weight of the error vector is t or smaller. This holds independently of which codeword $\mathbf{c} \in \mathcal{C}$ was transmitted.

Proposition 2.1. A code $\mathcal{C}$ with minimum distance $d_{\min}$ can correct $t \leq \lfloor \frac{d_{\min}-1}{2} \rfloor$ errors[1].

Before giving the proof, Fig. 2.1 illustrates the concepts. If each codeword is surrounded by a sphere of radius t, then any error sequence of weight t or less, will cause the received sequence to be decoded correctly. For successful decoding, the spheres cannot overlap. If $d_{\min}$ is odd, then the radius of this sphere is $t = \frac{1}{2}(d_{\min} - 1)$. If $d_{\min}$ is even, then there is a midway point that cannot be successfully decoded, and $t = \frac{1}{2}(d_{\min} - 2)$.

Proof Let $\mathbf{y} = \mathbf{c} + \mathbf{e}$ and assume $\mathrm{wt}_{\mathrm{H}}(\mathbf{e}) = s \leq t$, that is s errors have occurred. Claim: $\mathbf{y}$ is closer to $\mathbf{c}$ than to any other codeword. Assume the opposite, that is, there exists a $\mathbf{w} \in \mathcal{C}$ with $\mathbf{w} \neq \mathbf{c}$ such that $d(\mathbf{w}, \mathbf{y}) < d(\mathbf{c}, \mathbf{y})$. If so,

$$\begin{aligned} d(\mathbf{c}, \mathbf{w}) &\leq d(\mathbf{c}, \mathbf{y}) + d(\mathbf{y}, \mathbf{w}) && \text{Triangle inequality} \\ &\leq s + s && d(\mathbf{w}, \mathbf{y}) < d(\mathbf{c}, \mathbf{y}) = s \text{ by assumption} \\ &\leq 2t && \text{assumed } s \leq t \\ &\leq d_{\min} - 1. \end{aligned}$$

The last statement is a contradiction, because the minimum distance of the code is $d_{\min}$. □

The code in Example 1.1.2 has $d_{\min} = 7$, so it can correct 3 errors. The code in Example 1.1.3 has $d_{\min} = 3$, so it can can correct 1 error.

SSQ 2.2. Consider the nonlinear code

$$\mathcal{C} = \{(0,0,0,0,0,0), (0,0,1,1,1,1), (1,1,1,0,0,0), (1,1,0,1,1,1)\}$$

[1] Alternatively, $t < \frac{d_{\min}}{2}$, since $d_{\min}$ is an integer

What is the minimum distance $d_{\min}$ of this code? How many errors can $\mathcal{C}$ correct?

2.2 Linear Block Codes

2.2.1 Definition

Recall that a vector subspace $\mathcal{V}$ is a vector space which is a subset of $\mathbb{F}_q^n$. For $\mathbf{x}, \mathbf{y} \in \mathcal{V}$, it is closed under addition and multiplication by a scalar.

Definition 2.7. An (n, k) *linear block code* $\mathcal{C}$ is a k-dimensional subspace of a vector space $\mathbb{F}_q^n$.

The number of codewords is $M = q^k$. The dimension of the code $\mathcal{C}$ is k, and its blocklength[2] is n. The code rate is $R = \frac{1}{n} \log M = \frac{k}{n} \log q$ bits. It is common to write (n, k) to indicate a linear block code with block length n and dimension k. A code with minimum distance d may be written (n, k, d).

Since a linear code $\mathcal{C}$ is itself a vector subspace, the following properties hold:

- The sum of any two codewords is a codeword. That is, $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ means that $\mathbf{c}_1 + \mathbf{c}_2 \in \mathcal{C}$.
- The all-zeros vector is a codeword. That is, $\mathbf{0} = [0, 0, \ldots, 0] \in \mathcal{C}$ since the subspace must include the all-zeros vector.
- Scalar multiplication of a codeword is a codeword: $a \cdot \mathbf{c} \in \mathcal{C}$ for any $a \in \mathbb{F}$.

For a binary $q = 2$ code, $M = 2^k$, and its code rate R is $\frac{1}{n} \log M = \frac{k}{n}$. Since $k \leq n$, $R \leq 1$. A high-rate code can carry a large amount of information, but has low error-correction capability, compared to a low rate code, which carries less information but may have high error-correction capability.

2.2.2 Generator Matrix G

Since a linear block code $\mathcal{C}$ is a k-dimensional vector space, this space can be spanned by a set of k linearly independent basis vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k$, where,

$$\mathbf{g} = [g_1, g_2, \ldots, g_n] \tag{2.21}$$

is a row vector representing a point in $\mathbb{F}_q^n$. A vector $\mathbf{c}$ is a codeword if it can be formed as a linear combination of the basis vectors,

$$\mathbf{c} = u_1 \mathbf{g}_1 + u_2 \mathbf{g}_2 + \cdots + u_k \mathbf{g}_k. \tag{2.22}$$

[2]It is common to call n the length of the code, since this is the number of symbols — even in Matlab, the function `length` gives the number of elements in a vector. But for a point $\mathbf{x}$ in the n-dimensional real space $\mathbb{R}^n$, the *length* of $\mathbf{x}$ is its distance from the origin, not n. To reduce confusion, let us use "block length."

where $u_i \in \mathbb{F}_q$ for $i = 1, 2, \ldots, k$.

The symbols $u_i \in \mathbb{F}_q$ are called *information symbols* and can be written as a vector

$$\mathbf{u} = [u_1, u_2, \ldots, u_k]. \tag{2.23}$$

This represents the information that is to be transmitted over the unreliable channel. The mapping from information to codewords can be written in matrix form as:

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \tag{2.24}$$

where $\mathbf{G}$ is a k-by-n matrix called a *generator matrix*:

$$\mathbf{G} = \begin{bmatrix} — & \mathbf{g}_1 & — \\ — & \mathbf{g}_2 & — \\ & \vdots & \\ — & \mathbf{g}_k & — \end{bmatrix}. \tag{2.25}$$

If the k vectors $\mathbf{g}_1, \ldots, \mathbf{g}_k$ are linearly independent, then $\mathbf{G}$ is full rank. The entire codebook $\mathcal{C}$ can be found by multiplying all possible sequences $\mathbf{u} \in \mathbb{F}_q^k$ by $\mathbf{G}$.

Example 2.7. Find the code parameters n, k, R, M and the codebook $\mathcal{C}$ for the binary $q = 2$ code with generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{2.26}$$

The code dimension is $k = 3$ and the block length is $n = 5$, the code rate is $R = k/n = 3/5$, and the number of codewords is $M = 2^k = 8$. The codebook $\mathcal{C}$ is shown in the right column of the table below, obtained from $\mathbf{uG}$ for all $\mathbf{u} = (u_1, u_2, u_3) \in \mathbb{F}_2^3$:

information $\mathbf{u}$	codeword $\mathbf{c}$
$[0, 0, 0]$	$[0, 0, 0, 0, 0]$
$[1, 0, 0]$	$[1, 0, 0, 1, 0]$
$[0, 1, 0]$	$[0, 1, 0, 0, 1]$
$[1, 1, 0]$	$[1, 1, 0, 1, 1]$
$[0, 0, 1]$	$[0, 0, 1, 1, 1]$
$[1, 0, 1]$	$[1, 0, 1, 0, 1]$
$[0, 1, 1]$	$[0, 1, 1, 1, 0]$
$[1, 1, 1]$	$[1, 1, 1, 0, 0]$

It is easy to verify that the sum of any two codewords is also a codeword. For example the sum of the last two codewords $[0\ 1\ 1\ 1\ 0] + [1\ 1\ 1\ 0\ 0]$ is the second codeword $[1\ 0\ 0\ 1\ 0]$.

Example 2.8. Find all codewords of the code defined over $\mathbb{F}_3$, with generator matrix:

$$\begin{bmatrix} 1 & 0 & 1 & 2 & 0 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix}.$$

The codewords are given by $\mathbf{uG}$ where $\mathbf{u} = [u_1, u_2] \in \{0, 1, 2\}^2$:

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 2 \\ 0 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 1 & 0 & 1 \\ 2 & 0 & 2 & 1 & 0 \\ 2 & 1 & 2 & 0 & 2 \\ 2 & 2 & 2 & 2 & 1 \end{array}$$

SSQ 2.3. Consider a code with the following generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

What is the dimension of this code? What is the block length of this code? How many codewords does the code have?

2.2.3 Parity-Check Matrix H

A parity-check matrix $\mathbf{H}$ specifies constraints that all codewords $\mathbf{c} \in \mathcal{C}$ must satisfy, and is an alternative method to describe a linear block code. First, a parity check is described.

Definition 2.8. A *parity check* for a code with k-by-n generator matrix $\mathbf{G}$ is a 1-by-n vector $\mathbf{h}$ that satisfies:

$$\mathbf{G} \cdot \mathbf{h}^{\mathrm{t}} = \mathbf{0}, \tag{2.27}$$

where $\mathbf{0}$ is the k-by-1 all-zeros column vector.

Example 2.9. Consider a dimension $k = 2$ code with $\mathbf{G}$ given below, and $\mathbf{h}_1 = (1, 1, 0, 1, 0)$. Then,

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}}_{\mathbf{G}} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.28}$$

can be confirmed to be equal to $[0\ 0\ 0]^{\mathrm{t}}$ and $\mathbf{h}$ is a parity check for this code.

Definition 2.9. For a block length n code with dimension k, a *parity-check matrix* $\mathbf{H}$ is a matrix with n columns and $n-k$ linearly independent parity checks in its rows. Wikipedia: Parity check matrix

The parity-check matrix can be represented as:

$$\mathbf{H} = \begin{bmatrix} — & \mathbf{h}_1 & — \\ — & \mathbf{h}_2 & — \\ & \vdots & \\ — & \mathbf{h}_m & — \end{bmatrix}, \tag{2.29}$$

where $m \geq (n-k)$. $\mathbf{H}$ has $n-k$ linearly independent parity check vectors. If $m = n-k$, then all the parity check vectors are linearly independent. If $\mathbf{H}$ has more than $n-k$ rows, then at least one row is linearly dependent. Since each row of $\mathbf{H}$ is a parity check:

$$\mathbf{G}\mathbf{H}^{\mathrm{t}} = \mathbf{0}, \tag{2.30}$$

where $\mathbf{0}$ is a $k \times m$ all-zeros matrix.

Example 2.10. Continuing Example 2.9, it is easy to verify that $\mathbf{h}_2 = (0,1,0,0,1)$ and $\mathbf{h}_3 = (1,1,1,0,1)$ are also parity checks. Since the three parity checks $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ are linearly independent, they can be used to form a parity-check matrix $\mathbf{H}$, which satisfies (2.30):

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}^{\mathrm{t}}}_{\mathbf{H}^{\mathrm{t}}} = \mathbf{0}, \tag{2.31}$$

where $\mathbf{0}$ is the 2×3 all-zeros matrix.

The following proposition shows that a parity check matrix can be used to test whether a given sequence is a codeword or not.

Proposition 2.2. A sequence $\mathbf{x}$ is a codeword if and only if $\mathbf{x}\mathbf{H}^{\mathrm{t}} = \mathbf{0}$.

Proof Assume $\mathbf{x}$ is a codeword. There is some $\mathbf{u}$ such that $\mathbf{x} = \mathbf{u}\mathbf{G}$. Then, $\mathbf{x}\mathbf{H}^{\mathrm{t}} = \mathbf{u}\mathbf{G}\mathbf{H}^{\mathrm{t}} = \mathbf{x}\mathbf{0} = \mathbf{0}$. Conversely, if $\mathbf{x}$ is not a codeword it can be written as $\mathbf{x} = \mathbf{u}\mathbf{G} + \mathbf{e}$, where $\mathbf{e}$ is a vector which is not a codeword. Multiplying by $\mathbf{H}^{\mathrm{t}}$, we have $\mathbf{x}\mathbf{H}^{\mathrm{t}} = \mathbf{e}\mathbf{H}^{\mathrm{t}}$. By assumption, $\mathbf{e}$ is not a codeword, so $\mathbf{e}\mathbf{H}^{\mathrm{t}} \neq 0$ and we have $\mathbf{x}\mathbf{H}^{\mathrm{t}} \neq \mathbf{0}$. (If $\mathbf{H}$ contains an all-zero column, the corresponding symbol is independent of the codeword.) □

SSQ 2.4. Consider the code $\mathcal{C}$ with parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{2.32}$$

Is $[0,0,1,0,0,1]$ a codeword of $\mathcal{C}$? Is $[1,0,1,0,1,0]$? Is $[1,0,0,1,1,0]$?

2.2.4 Minimum Distance

The minimum distance of an error correcting code was given in Definition 2.6. For a linear block code, the minimum distance is the minimum of the weight of non-zero codewords. That is,

$$d_{\min} = \min_{\mathbf{c}\in\mathcal{C},\mathbf{c}\neq 0} w(\mathbf{c}) \tag{2.33}$$

This can be seen by observing:

$$d_{\min} = \min_{\mathbf{x},\mathbf{y}\in\mathcal{C},\mathbf{x}\neq\mathbf{y}} d(\mathbf{x},\mathbf{y}) = \min_{\mathbf{c}\in\mathcal{C},\mathbf{c}\neq 0} d(\mathbf{c},0) = \min_{\mathbf{c}\in\mathcal{C},\mathbf{c}\neq 0} w(\mathbf{c}) \tag{2.34}$$

using $\mathbf{c} = \mathbf{x} - \mathbf{y}$.

In this way, the minimum distance of a code may be found from the minimum of the weights of the codewords, rather than computing the distance between all pairs of codewords.

Example 2.11. The minimum distance of the code in Example 2.7, can be found. The minimum distance is 2, since there are two nonzero codewords of weight two, and none of weight one.

For binary codes, linearly dependent columns sum to the all-zeros column vector.

Proposition 2.3. Let $\mathbf{H}$ be a parity check matrix for a code $\mathcal{C}$. Then the minimum distance of $\mathcal{C}$ is equal to the minimum number of linearly dependent columns of $\mathbf{H}$.

Example 2.12. Find the minimum distance of the code with the following parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.35}$$

There are no two columns in $\mathbf{H}$ which sum to 0, so the minimum distance is greater than 2. Columns 1, 2 and 3 sum to $[0\ 1\ 1]^{\mathrm{t}}$, so they are not linearly dependent. But columns 2, 3 and 4 sum to $[0\ 0\ 0]^{\mathrm{t}}$, so they are linearly dependent and the minimum distance of this code is 3.

SSQ 2.5. What is the minimum distance of the code with the following

parity-check matrix:

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

2.3 Systematic Forms

2.3.1 Systematic Generator Matrix

In a code's *systematic form*, the k information symbols $u_1, u_2, \ldots, u_k$ usually appear in positions 1 to k of the codeword. The remaining $m = n - k$ symbols appear in positions $k+1$ to n and are called *parity symbols*, so the codeword $\mathbf{c}$ is:

$$\mathbf{c} = [u_1 \; u_2 \; \ldots \; u_k \; p_1 \; p_2 \; \ldots \; p_m] \tag{2.36}$$

In this case, the generator matrix in systematic form is:

$$\mathbf{G} = \left[\begin{array}{c|c} \mathbf{I}_k & \mathbf{A} \end{array} \right] \tag{2.37}$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{A}$ is a $k \times m$ matrix over $\mathbb{F}_q$. Any linear block code has a systematic form, although the systematic bits are not necessarily in the first k positions. In this case, the bit positions can be permuted so that the information bits are in the first k positions. This corresponds to swapping columns of the generator matrix.

A code $\mathcal{C}$ has many possible bases and thus many possible generator matrices — two distinct matrices $\mathbf{G}_1$ and $\mathbf{G}_2$ may both generate a code $\mathcal{C}$, although two distinct generator matrices will give distinct mappings from information $\mathbf{u}$ to codewords $\mathbf{c}$. However, a code $\mathcal{C}$ has only one systematic generator matrix.

A generator matrix for a code $\mathcal{C}$ can be transformed to another generator matrix for the same code $\mathcal{C}$ using standard row operations:

1. For binary codes, replace a row with that row plus another row. (For non-binary codes, replace a row with that row plus another row times a non-zero constant),
2. Two rows may be exchanged.

To obtain the systematic generator matrix, these operations are performed until the k-by-k identity matrix appears on the left.

Example 2.13. Consider a non-systematic generator $\mathbf{G}_{\text{non}}$:

$$\mathbf{G}_{\text{non}} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.38}$$

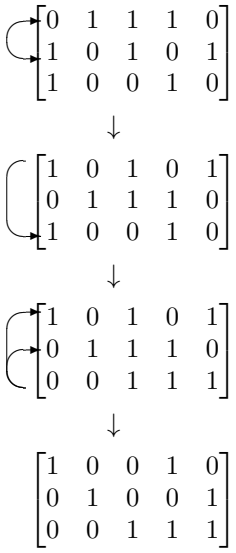


Figure 2.2: Row operations to find the generator matrix in systematic form.

This can be transformed into a systematic generator matrix using row operations as shown in Fig. 2.2. The systematic matrix for the same code is:

$$\mathbf{G}_{\text{sys}} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.39}$$

The three information bits are the first three positions of the codeword. Note that $\mathbf{G}_{\text{non}}$ and $\mathbf{G}_{\text{sys}}$ are two generator matrices for the same code.

SSQ 2.6. Find the systematic generator matrix for a code with generator matrix:

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.40}$$

2.3.2 Systematic Parity-Check Matrix

The parity-check matrix also has a systematic form. If $\mathbf{H}$ is an $m \times n$ matrix, then the systematic parity check form is:

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & | & \mathbf{I}_m \end{bmatrix}. \tag{2.41}$$

Row operations may be applied to a parity check matrix $\mathbf{H}$ to find the systematic form. Given an arbitrary matrix $\mathbf{H}$, it may not be possible to put the matrix into systematic form, unless column swaps are allowed.

Example 2.14. Put the following binary parity check matrix $\mathbf{H}$ in systematic form using row operations.

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

Replace row 3 with the sum of row 2 and row 3. Replace row 2 with the sum of row 1 and row 2, giving:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.42}$$

which is the systematic form.

The systematic forms can be used to convert between the parity-check matrix and generator matrix for a code. In particular, if a code parity-check matrix is $\mathbf{H} = [\mathbf{A} \mid \mathbf{I}_k]$, the generator matrix $\mathbf{G}$ for the code is:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_k & | & -\mathbf{A}^{\mathrm{t}} \end{bmatrix}. \tag{2.43}$$

For binary codes, $-\mathbf{A}^{\mathrm{t}} = \mathbf{A}^{\mathrm{t}}$.

Example 2.15. Find the generator matrix for the code in Example 2.14. Since $\mathbf{H} = [\mathbf{A} \mid \mathbf{I}_k]$ as in (2.42), the generator matrix is given by:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_k & -\mathbf{A}^{\mathrm{t}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{2.44}$$

Example 2.16. Put the following ternary $q = 3$ generator matrix $\mathbf{G}$ in systematic form and find the corresponding parity-check matrix $\mathbf{H}$.

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & 1 & 2 & 0 \\ 2 & 2 & 0 & 2 & 2 \end{bmatrix} \tag{2.45}$$

Replace the second row with the sum of the first and second row:

$$\begin{bmatrix} 1 & 2 & 1 & 2 & 0 \\ 0 & 1 & 1 & 1 & 2 \end{bmatrix} \tag{2.46}$$

And the replace the first row with the sum of the first and second row:

$$\begin{bmatrix} 1 & 0 & 2 & 0 & 2 \\ 0 & 1 & 1 & 1 & 2 \end{bmatrix} \tag{2.47}$$

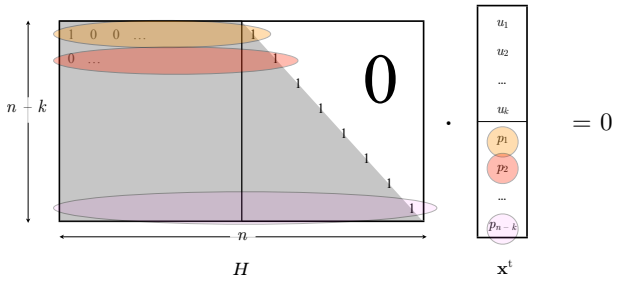


Figure 2.3: Systematic encoding by back-substitution.

We have a generator matrix in systematic form. The parity check matrix in systematic form is:

$$\mathbf{H} = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.48}$$

2.3.3 Encoding Using H

Encoding can be performed using the parity-check matrix $\mathbf{H}$ written as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & | & \mathbf{T} \end{bmatrix} \tag{2.49}$$

where $\mathbf{T}$ is a $m \times m$ lower triangular matrix with no zeros on the diagonal. This encoding does not require the generator matrix.

Encoding of data $\mathbf{u}$ to codeword $\mathbf{c}$ can be performed by writing $\mathbf{c} = \begin{bmatrix} \mathbf{u} & \mathbf{p} \end{bmatrix}$; this encoding is systematic. To find the unknown parity $\mathbf{p}$, recognize that any codeword $\mathbf{c}$ must satisfy $\mathbf{H}\mathbf{c}^{\mathrm{t}} = \mathbf{0}$ (Proposition 2.2):

$$\begin{bmatrix} \mathbf{P} & | & \mathbf{T} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ p_1 \\ p_2 \\ \vdots \\ p_m \end{bmatrix} = \mathbf{0} \tag{2.50}$$

This can be solved row-by-row, starting with the top row, first finding p_1, then $p_2, p_3, \ldots$ in order, as shown in the following example. This procedure is sometimes called back substitution, see Fig. 2.3.

Example 2.17. Encode information $\mathbf{u} = [1, 1, 0]$ to the systematic codeword of the code with parity-check matrix $\mathbf{H}$ in lower triangular form:

$$\mathbf{H} = \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{array}\right]. \tag{2.51}$$

This can be done by solving:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \mathbf{0}. \tag{2.52}$$

The top row of the above equality is $1+0+0+p_1 = 0$, which means that $p_1 = 1$. The second row is $0 + 0 + 0 + p_1 + p_2 = 0$; since $p_1 = 1$, we have $p_2 = 1$. The last row is $1 + 1 + 0 + p_1 + 0 + p_3 = 0$, so $p_3 = 1$, giving:

$$[1, 1, 0, 1, 1, 1] \tag{2.53}$$

which is the systematic codeword.

If the parity-check matrix is in nearly lower-triangular form, sometimes called approximate-lower triangular (ALT) form, then efficient encoding is still possible. This is discussed for LDPC codes, see Section 4.4.

2.4 Important Linear Block Codes

This section describes some important linear block codes

2.4.1 Repeat Code

Definition 2.10. For $n \geq 1$, the *repeat code* is an $(n, 1, n)$ code with generator matrix:

$$\mathbf{G}_{\text{repeat}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2.54}$$

In the binary case, the codebook has just two codewords, $\mathcal{C} = \{(0\,0\,\ldots\,0), (1\,1\,\ldots\,1)\}$. The rate of this code is $R = \frac{1}{n}$ and the minimum distance is $d_{\min} = n$. This is a low-rate code with high minimum distance.

2.4.2 Single-Parity Check Code

Definition 2.11. For $n \geq 2$, the *single parity-check code* is an $(n, n-1, 2)$ code with parity-check matrix:

$$\mathbf{H}_{\text{SPC}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{2.55}$$

A binary single-parity check code or SPC code consists of all binary sequences of length n $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that sum to 0, modulo 2, that is, they have even weight:

$$\{\mathbf{x} : \sum_{i=1}^{n} x_i = 0 \bmod 2\} \tag{2.56}$$

For example if $n = 4$, then the binary SPC codebook consists of all length 4 sequences with even weight:

$$\mathcal{C} = \{0000, 1100, 1010, 1001, 0110, 0101, 0011, 1111\}. \tag{2.57}$$

The rate is $R = \frac{n-1}{n}$ and the minimum distance is $d_{\min} = 2$. This is a high-rate code with low minimum distance.

2.4.3 Hamming Code

Hamming codes are high-rate binary codes with a minimum distance of three. Extended Hamming codes have minimum distance of four.

Definition 2.12. For $m \geq 2$, a binary *Hamming code* is a $(2^m - 1, 2^m - m - 1, 3)$ code whose parity check matrix consists of all non-zero binary m vectors as columns.

A Hamming code has length $n = 2^m - 1$, for $m = 2, 3, 4, \ldots$. The number of information symbols is $k = 2^m - 1 - m$. The Hamming code has a parity-check matrix of size $m \times 2^m - 1$.

Example 2.18. The parity check matrix for the Hamming code with $m = 3$ is:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.58}$$

which shows all non-zero binary vectors in columns.

Write the codeword as $\mathbf{c} = [u_1, u_2, u_3, u_4, p_1, p_2, p_3]^{\mathrm{t}}$, where u_i are systematic bits and p_i are parity bits. From $\mathbf{Hc} = \mathbf{0}$, the parity-check equations can be written as:

$$u_1 + u_2 + u_3 + p_1 = 0 \tag{2.59}$$
$$u_1 + u_2 + u_4 + p_2 = 0 \tag{2.60}$$
$$u_1 + u_3 + u_4 + p_3 = 0 \tag{2.61}$$

Given four information bits u_1, u_2, u_3, u_4, the parity bits can be found as $p_1 = u_1 + u_2 + u_3$ since $u_i = -u_i$ in modulo-2 arithmetic. This corresponds to Example 1.1.3.

The Hamming code has $d_{\min} = 3$, which can be shown using Proposition 2.3. Since the columns of $\mathbf{H}$ consist of all non-zero binary vectors, there are no two columns which sum to the zero column $\mathbf{0}$, so the minimum distance must be greater than 2. But for any two column vectors, there is always a third equal to their sum, so there exist 3 column vectors that sum to $\mathbf{0}$. So, there are 3 linearly dependent column vectors and the minimum distance of the Hamming code is 3.

Extended Codes Any code can be extended. If a code $\mathcal{C}$ has odd minimum distance d, then the extended code $\mathcal{C}'$ will have minimum distance $d+1$. The parity-check matrix of $\mathcal{C}'$ is obtained by adding the all-zeros column vector to the parity-check matrix of $\mathcal{C}$, then adding the all-ones row vector.

Specifically for Hamming codes, an *extended Hamming code* of length $n = 2^m$ exists for $m = 2, 3, 4, 5, \ldots$. The extended Hamming code has $d_{\min} = 4$.

Example 2.19. The parity check matrix for the $(8, 4, 4)$ extended Hamming code with $m = 3$ is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.62}$$

The elements added to the Hamming code to form the extended Hamming code are colored.

SSQ 2.7. What are the code rates R of the Hamming codes with $m = 4, 5$ and 6?

2.4.4 First-Order Reed-Muller Codes

Reed-Muller codes are family of error-correcting codes. For block length $n \leq 32$, they are among the best-known codes. For $m \geq 1$, let $\mathbf{v}_0 = [1, 1, \cdots, 1]$ be the all-ones vector of block length 2^m. Let $\mathbf{V}$ be the m-by-2^m matrix consisting of all binary m vectors as columns. Let row i of this matrix be $\mathbf{v}_i$, $i = 1, \ldots, m$. For example for $m = 3$:

$$\mathbf{v}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \text{ and}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Wikipedia: Reed–Muller code

Definition 2.13. For $m \geq 1$, a *first-order Reed-Muller code* is a $(2^m, m+1)$ block code where the first row of the generator matrix is the all-ones vector and rows 2 to $m+1$ consist of all binary m vectors as columns.

Example 2.20. For $m = 2$, the $(4, 3)$ first-order Reed-Muller code has generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.63}$$

For $m = 3$, the $(8, 4)$ first-order Reed-Muller code has generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.64}$$

The minimum distance of a first-order Reed-Muller code is 2^{m-1}. In the first-order Reed-Muller code's generator matrix, each row has weight 2^{m-1} except one row of weight 2^m.

2.4.5 Reed–Muller Codes of Higher Order

Reed-Muller codes can be generalized to higher orders. First, for two binary vectors $\mathbf{v}$, and $\mathbf{u}$, let their *binary product* denoted $\mathbf{v} \cdot \mathbf{u}$ be the symbol-by-symbol product. For example, $[0, 0, 1, 1] \cdot [0, 1, 0, 1] = [0, 0, 0, 1]$.

Recall the definition of $\mathbf{V}$ with row vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$. Consider choosing r of these m vectors, and then taking the binary product of these r vectors. In total, there are $\binom{m}{r}$ possible products.

Definition 2.14. Let $0 \leq r \leq m$. An order r Reed Muller with block length $n = 2^m$ denoted $\mathrm{RM}(r, m)$, has generator matrix consisting of $\mathbf{v}_0$ and the binary products of up to r of the $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$.

An (r, m) Reed-Muller code has block length 2^m and dimension k given by:

$$k = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r} \tag{2.65}$$

The rows of its generator matrix is given by binary vectors of order $0, 1, \ldots, r$. The minimum distance is 2^{m-r}.

The block length is $n = 2^m$, for an order m. For $m = 4$, the following gives

the vectors for each order $r = 0, 1, 2, 3, 4$.

$$\left.\begin{aligned} \mathbf{1} &= 1111\ 1111\ 1111\ 1111 \end{aligned}\right\} \quad \text{order 0}$$

$$\left.\begin{aligned} \mathbf{v}_1 &= 0101\ 0101\ 0101\ 0101 \\ \mathbf{v}_2 &= 0011\ 0011\ 0011\ 0011 \\ \mathbf{v}_3 &= 0000\ 1111\ 0000\ 1111 \\ \mathbf{v}_4 &= 0000\ 0000\ 1111\ 1111 \end{aligned}\right\} \quad \text{order 1}$$

$$\left.\begin{aligned} \mathbf{v}_1 \cdot \mathbf{v}_2 &= 0001\ 0001\ 0001\ 0001 \\ \mathbf{v}_1 \cdot \mathbf{v}_3 &= 0000\ 0101\ 0000\ 0101 \\ \mathbf{v}_1 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0101\ 0101 \\ \mathbf{v}_2 \cdot \mathbf{v}_3 &= 0000\ 0011\ 0000\ 0011 \\ \mathbf{v}_2 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0011\ 0011 \\ \mathbf{v}_3 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0000\ 1111 \end{aligned}\right\} \quad \text{order 2}$$

$$\left.\begin{aligned} \mathbf{v}_1 \cdot \mathbf{v}_2 \cdot \mathbf{v}_3 &= 0000\ 0001\ 0000\ 0001 \\ \mathbf{v}_1 \cdot \mathbf{v}_2 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0001\ 0001 \\ \mathbf{v}_1 \cdot \mathbf{v}_3 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0000\ 0101 \\ \mathbf{v}_2 \cdot \mathbf{v}_3 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0000\ 0011 \end{aligned}\right\} \quad \text{order 3}$$

$$\left.\begin{aligned} \mathbf{v}_1 \cdot \mathbf{v}_2 \cdot \mathbf{v}_3 \cdot \mathbf{v}_4 &= 0000\ 0000\ 0000\ 0001 \end{aligned}\right\} \quad \text{order 4}$$

To construct a Reed-Muller code of order r, the generator vectors are the products of order r and lower.

Example 2.21. The RM$(4, 2)$ is a (16,11,4) code with generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1111\ 1111\ 1111\ 1111 \\ 0101\ 0101\ 0101\ 0101 \\ 0011\ 0011\ 0011\ 0011 \\ 0000\ 1111\ 0000\ 1111 \\ 0000\ 0000\ 1111\ 1111 \\ 0001\ 0001\ 0001\ 0001 \\ 0000\ 0101\ 0000\ 0101 \\ 0000\ 0000\ 0101\ 0101 \\ 0000\ 0011\ 0000\ 0011 \\ 0000\ 0000\ 0011\ 0011 \\ 0000\ 0000\ 0000\ 1111 \end{bmatrix} \tag{2.66}$$

Several codes can be seen as instances of Reed-Muller codes, making RM codes a generalization of other codes:

- RM(0,m) codes are Repetition code of length $n = 2^m$, code rate $R = 1/n$ and $d_{\min} = n$.
- For $m = 1, 3, 5, \ldots,$, certain RM codes have $k = n/2$ and are self-dual codes.

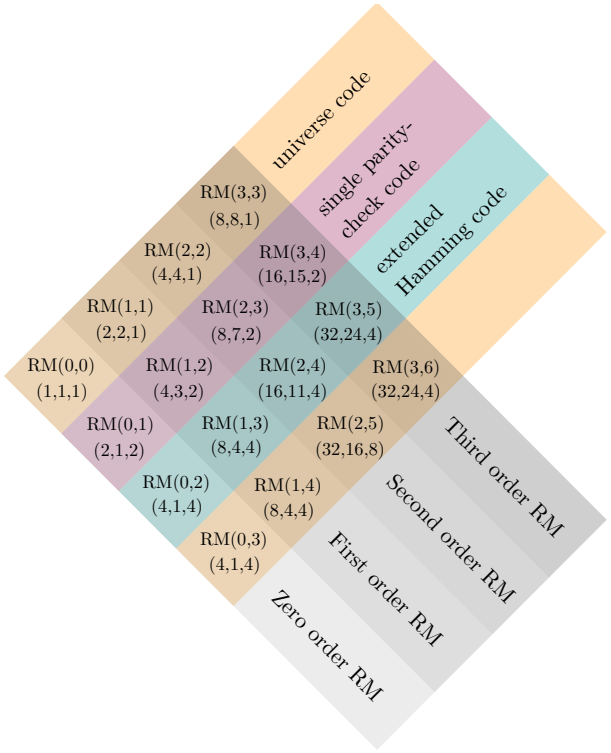


Figure 2.4: Connections between Reed-Muller codes and other codes. The universe code is the set of all binary sequences $\mathbb{F}_2^n$.

- RM($m-1$,m) codes are single-parity check code of length 2^m, code rate $R = \frac{n-1}{n}$ and minimum distance $d_{\min} = 2$.

- RM($m-2$,m) codes are extended Hamming codes of length 2^m, and minimum distance $d_{\min} = 4$.

These relationships are illustrated in Fig. 2.4.

2.5 Weight Distribution (Distance Spectrum)

The weight distribution of a code is the number of codewords for each weight w. The weight distribution is sometimes called distance spectrum.

Definition For a block length n linear code, the weight distribution is a sequence $A_0, A_1, A_2, \cdots, A_n$ where A_w is the number of codewords with weight w.

It is convenient to write a weight distribution polynomial (also called weight

enumerator polynomial) $A(z)$:

$$A(z) = \sum_{w=0}^{n} A_w z^w. \tag{2.67}$$

Here z is just a placeholder variable or dummy variable. For any linear code, $A_0 = 1$, that is there is one all-zeros codeword. After A_0, the next non-zero term is at the minimum distance, that is $A_{d_{\min}}$.

The following is a list of all codewords of the $(7, 4)$ Hamming code, and their weights:

codeword $\mathbf{x}$	$\mathrm{wt_H}(\mathbf{x})$	codeword $\mathbf{x}$	$\mathrm{wt_H}(\mathbf{x})$
0 0 0 0 0 0 0	0	1 1 0 1 0 0 1	4
1 1 1 0 0 0 0	3	0 0 1 1 0 0 1	3
1 0 0 1 1 0 0	3	0 1 0 0 1 0 1	3
0 1 1 1 1 0 0	4	1 0 1 0 1 0 1	4
0 1 0 1 0 1 0	3	1 0 0 0 0 1 1	3
1 0 1 1 0 1 0	4	0 1 1 0 0 1 1	4
1 1 0 0 1 1 0	4	0 0 0 1 1 1 1	4
0 0 1 0 1 1 0	3	1 1 1 1 1 1 1	7

From this, $A_0 = 1, A_3 = 7, A_4 = 7, A_7 = 1$ and all other $A_w = 0$. The weight distribution polynomial is:

$$A(z) = 1 + 7z^3 + 7z^4 + z^7 \tag{2.68}$$

2.6 Exercises

2.1 Consider a non-linear code $\mathcal{C}_{\mathrm{NL}}$ with four codewords:

$$\begin{aligned}
\mathbf{c}_1 &= [1, 0, 0, 1, 0] \\
\mathbf{c}_2 &= [0, 1, 0, 0, 1] \\
\mathbf{c}_3 &= [1, 0, 1, 0, 1] \\
\mathbf{c}_4 &= [0, 1, 1, 1, 0]
\end{aligned}$$

(a) Find the minimum distance of the non-linear code $\mathcal{C}_{\mathrm{NL}}$ above.
(b) Add the minimal number of codewords to $\mathcal{C}_{\mathrm{NL}}$ to form a binary linear code $\mathcal{C}_{\mathrm{L}}$. Find the systematic generator matrix for $\mathcal{C}_{\mathrm{L}}$. What is the dimension and minimum distance of $\mathcal{C}_{\mathrm{L}}$?

2.2 Using encoding by back-substitution, for the code with parity-check matrix:

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}$$

encode the information sequences $\mathbf{u}_1 = (1, 0, 1)$ and $\mathbf{u}_2 = (0, 1, 0)$ to the corresponding codewords $\mathbf{c}_1$ and $\mathbf{c}_2$.

2.3 Find the systematic generator for the code with the parity check matrix $\mathbf{H}$:

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

Note that $\mathbf{H}$ is not full rank.

2.4 What is the minimum distance of the code with the following parity-check matrix? How many codewords does the code have?

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

2.5 For a code over $\mathbb{F}_5$ with generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 3 & 1 \\ 0 & 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 & 3 \end{bmatrix},$$

find the parity check matrix $\mathbf{H}$ in systematic form. For $\mathbb{F}_5$, use modulo 5 addition and multiplication.

2.6 Consider a code $\mathcal{C}$ with generator matrix:

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.72}$$

(a) Write the systematic generator matrix for $\mathcal{C}$
(b) How many *distinct* codewords does $\mathcal{C}$ have? Write all the codewords.
(c) What is the block length, dimension and minimum distance of $\mathcal{C}$?
(d) How did you find the minimum distance?

2.7 Prove the following statements.

(a) Let $\mathbf{x}$ be a sequence with even Hamming weight, and let $\mathbf{y}$ be a sequence with Hamming weight 1. Show that $\mathrm{d}(\mathbf{x}, \mathbf{y})$ is odd. Generalize this to $\mathbf{x}$ even Hamming weight and $\mathbf{y}$ odd Hamming weight.

(b) For a linear block code $\mathcal{C}$ over $\mathbb{F}_2$, prove that either (1) $\mathcal{C}$ has no codewords of odd weight, or (2) that half the codewords of $\mathcal{C}$ have odd weight.

(c) For a code $\mathcal{C}$ with parity-check matrix $\mathbf{H}$, its extended code $\mathcal{C}'$ has parity-check matrix $\mathbf{H}'$ obtained by adding one column of zeros and one row of ones:

$$\mathbf{H}' = \begin{bmatrix} & & & 0 \\ & \mathbf{H} & & \vdots \\ & & & 0 \\ 1 & \cdots & 1 & 1 \end{bmatrix}. \tag{2.75}$$

Let the minimum distance $d_{\min}$ of a code $\mathcal{C}$ be odd. Prove that the extended code $\mathcal{C}'$ has minimum distance $d_{\min} + 1$.

Chapter 3

Decoding Linear Block Codes

3.1 Principles of Decoding

3.1.1 Optimal Decoding

There are various types of decoding algorithms, that depend on the channel model and the code being decoded.

Maximum-likelihood (ML) decoding ML decoding is an optimal decoding metric. It selects the codeword that maximizes the likelihood $\Pr(\mathbf{y}|\mathbf{c})$, that is the output is $\widehat{\mathbf{c}}$:

$$\widehat{\mathbf{c}} = \arg\max_{\mathbf{c}\in\mathcal{C}} \Pr(\mathbf{y}|\mathbf{c}) \tag{3.1}$$

ML decoding is often regarded as optimal decoding. However, for general codes ML decoding has exponential complexity and is too complicated and lower-complexity algorithms are often used.

A posteriori probability (APP) decoding APP decoding outputs soft probabilities, say $r(c)$:

$$r(c) = \Pr(c_i = c|\mathbf{y}). \tag{3.2}$$

APP decoding is a component in other algorithms, where the soft output is passed to another decoder.

Maximum a posteriori (MAP) decoding MAP decoding is symbol-by-symbol (or bit-by-bit, in the binary case). It finds the code symbol c_i which maximizes $\Pr(c_i|\mathbf{y})$, that is $\widehat{c}_i$ is:

$$\widehat{c}_i = \arg\max_{c} \Pr(c_i|\mathbf{y}) \tag{3.3}$$

for $i = 1, 2, \ldots, n$. It gives the highest probability codeword *symbol* c_i, on a symbol-by-symbol basis, whereas ML decoding gives the mostly likely *codeword*. MAP decoding can be seen as applying a hard decision to the output as APP decoding.

Soft-input decoding vs Hard-input decoding A soft-input decoder accepts real values, usually the output of the AWGN channel or probabilities. A hard-input decoder for a binary code accepts only 0 and 1 as inputs (or more generally code symbols as inputs; erasure symbols may also be accepted as input). Hard inputs can be formed from a soft-output channels by making hard decisions symbol-by-symbol. However, discarding soft information by making hard decisions reduces the error-rate performance in general.

Bounded-distance decoding For a code with minimum distance $d_{\min}$, the error-correction capability is $t = \lfloor \frac{d_{\min}-1}{2} \rfloor$, as was shown in Proposition 2.1. Each codeword is at the center of a non-overlapping Hamming ball of radius t. In bounded-distance decoding, the decoder will output the codeword $\widehat{\mathbf{c}}$ if the received sequence $\mathbf{y}$ the Hamming distance between $\mathbf{y}$ and $\widehat{\mathbf{c}}$ is less than or equal to t. But, since spheres do not exactly cover the entire space $\mathbb{F}_q^n$ in general, a failure to decode event E occurs when the received sequence is outside of a sphere. The Berelkamp-Massey algorithm for decoding Reed-Solomon codes and BCH codes is an example of a bound-distance decoder.

3.1.2 Probabilistic Decoder Messages

Recall from Chapter 1 that channel models such as the BSC and AWGN channel are specified by a conditional probability distribution $p_{\mathsf{Y}|\mathsf{X}}(y|x)$. For an input sequence $\mathbf{x}$ and output sequence $\mathbf{y}$, these channels are memoryless, meaning:

$$p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p_{\mathsf{Y}|\mathsf{X}}(y_i|x_i) \tag{3.4}$$

While the channel output is $\mathbf{y}$, many decoding algorithms are probabilistic. Rather that working with y directly, instead consider probabilistic messages about y:

Probability Domain If x was transmitted, then we consider

$$r(x) = \Pr(\mathsf{X} = x|\mathsf{Y} = y), \tag{3.5}$$

Usually, x is binary, then there are two values $r(0)$ and $r(1)$. But since $r(0) + r(1) = 1$, we only need to know one of those two values. So there are two probability domains:

$$\text{P0 domain: } r(0) = \Pr(\mathsf{X} = 0|\mathsf{Y} = y) \text{ and} \tag{3.6}$$

$$\text{P1 domain: } r(1) = \Pr(\mathsf{X} = 1|\mathsf{Y} = y). \tag{3.7}$$

Log Domain We can also work in the log domain. Specifically, define:

$$R = \log \frac{r(0)}{r(1)} = \log \frac{\Pr(\mathsf{X}=0|\mathsf{Y}=y)}{\Pr(\mathsf{X}=1|\mathsf{Y}=y)} \tag{3.8}$$

It is possible to translate back to the probability domain by:

$$r(0) = \frac{e^R}{1+e^R} \text{ and } r(1) = \frac{1}{1+e^R} \tag{3.9}$$

Here, a lower case letter indicates the probability domain, and an upper case letter indicates the log domain.

For the BSC channel with error probability p, the channel output is $y \in \{0,1\}$. The log domain message L is:

$$L = \begin{cases} \log \frac{1-p}{p} & y=0 \\ \log \frac{p}{1-p} & y=1 \end{cases} \tag{3.10}$$

For $y=0$, this can be seen as $\Pr(\mathsf{X}=0|\mathsf{Y}=0) = 1-p$ and $\Pr(\mathsf{X}=1|\mathsf{Y}=0) = p$; similarly for $y=1$.

For the AWGN channel, the transmitter maps code bits c to transmitted symbols x as $\{0,1\} \to \{+1,-1\}$. Using Gaussian mean 0, variance σ^2 noise:

$$\Pr[\mathsf{Y}=y|\mathsf{X}=x] = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-x)^2}{2\sigma^2}} \tag{3.11}$$

Then, the channel output is y the log domain message L is given by:

$$L = \log e^{-\frac{(y-1)^2}{2\sigma^2}} - \log e^{-\frac{(y+1)^2}{2\sigma^2}} \tag{3.12}$$

$$L = \frac{2}{\sigma^2} y \tag{3.13}$$

That is, for the binary-input AWGN channel, convert from received symbols y to LLRs by multiplying by $\frac{2}{\sigma^2}$.

Log-Likelihood Ratio (LLR) The log-likelihood ratio (LLR) is:

$$R = \log \frac{\Pr(\mathsf{X}=0|\mathsf{Y}=y)}{\Pr(\mathsf{X}=1|\mathsf{Y}=y)}$$

Most commonly, $\Pr(\mathsf{X}=0) = \Pr(\mathsf{X}=1) = \frac{1}{2}$, and by Bayes rule, the LLR is equal to the log domain message.

3.1.3 Decoding the Repeat Code

Consider decoding the binary $(n,1)$ repeat code. The information is $x \in \{0,1\}$ and codeword $\{000\ldots0, 111\ldots1\}$ is transmitted over the channel. The channel output $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ is received. The decoder finds the APP estimate in the log domain:

$$Q = \log \frac{\Pr[\mathsf{X}=0|\mathsf{Y}=\mathbf{y}]}{\Pr[\mathsf{X}=1|\mathsf{Y}=\mathbf{y}]}$$

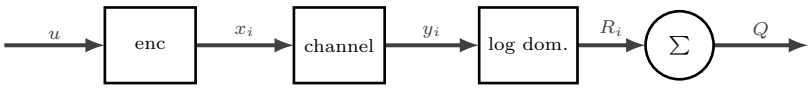


Figure 3.1: Block diagram for repeat code.

Here X and Y_j are random variables corresponding to x and y_j respectively. For the repeat code, this is computed as:

$$Q = \sum_{i=1}^{n} R_i \text{ where } R_i = \log \frac{\Pr[\mathsf{Y}_i = y_i | \mathsf{X} = 0]}{\Pr[\mathsf{Y}_i = y_i | \mathsf{X} = 1]} \tag{3.14}$$

as can be shown using the independence of the channel noise. The MAP estimate is a hard decision at 0:

$$\widehat{x} = \begin{cases} 0 & \text{if } Q \geq 0 \\ 1 & \text{if } Q < 0 \end{cases} \tag{3.15}$$

If $Q = 0$, in principle one may flip a coin to choose $\widehat{x}$, since $x = 0$ and $x = 1$ are equally likely. But in many cases, one may consistently choose one value, $x = 0$ as above.

Example 3.1. An $n = 3$ binary repeat code was transmitted over the BI-AWGN channel with $\sigma^2 = 0.8$, where the code symbols are mapped to channel inputs $\{0, 1\} \to \{+1, -1\}$. Find the APP and MAP estimates when

$$\mathbf{y} = [0.8, -1.4, 1.2]$$

is received.

Using (3.14), find

$$\begin{aligned} R_1 &= \log \frac{\Pr[\mathsf{Y}_i = y_i | \mathsf{X} = 0]}{\Pr[\mathsf{Y}_i = y_i | \mathsf{X} = 1]} = \frac{2}{\sigma^2} y_1 = 2.0 \\ R_2 &= -3.5 \\ R_3 &= 3.0 \\ \text{so } Q &= \sum_{i=1}^{n} R_i = 1.5 \end{aligned}$$

Using (3.9), $\Pr[\mathsf{X} = 0] = 0.818$ and $\Pr[\mathsf{X} = 1] = 0.182$, meaning that $x = 0$ has higher probability than $x = 1$. The MAP decision is $\widehat{x} = 0$ since $Q > 0$, that is, the symbol with highest probability is selected.

The probability domain decoding for the repeat code is:

$$q(0) = \frac{\prod_{j=1}^{n} r_j(0)}{\prod_{j=1}^{n} r_j(0) + \prod_{j=1}^{n} r_j(1)} \tag{3.16}$$

$$q(1) = \frac{\prod_{j=1}^{n} r_j(1)}{\prod_{j=1}^{n} r_j(0) + \prod_{j=1}^{n} r_j(1)}. \tag{3.17}$$

The probability domain operation multiplies the inputs $r_1(0)r_2(0)r_3(0)$ to find $q(0)$. However, because we require $q(0)+q(1)=1$, it is necessary to normalize, using the term written in the denominator. For $n=3$, the probability domain APP value is:

$$q(0) = \frac{r_1(0)r_2(0)r_3(0)}{r_1(0)r_2(0)r_3(0) + r_1(1)r_2(1)r_3(1)} \tag{3.18}$$

$$q(1) = \frac{r_1(1)r_2(1)r_3(1)}{r_1(0)r_2(0)r_3(0) + r_1(1)r_2(1)r_3(1)} \tag{3.19}$$

The MAP decision, or hard decision, is made using the APP values:

$$\widehat{x} = \begin{cases} 0 & \text{if } q(0) \geq 0.5 \\ 1 & \text{if } q(0) < 0.5 \end{cases} \tag{3.20}$$

3.1.4 Decoding of the Single-Parity Check Code

Consider decoding of the $(n, n-1)$ single-parity check code. n code symbols $x_1, x_2, \ldots, x_n$ are transmitted over a channel, where $x_1+x_2+\cdots+x_n=0$ is satisfied. The decoder finds the APP estimate $\Pr(\mathsf{X}_i|\mathbf{Y})$ in the probability domain. In the case of $n=3$ the codebook is $\mathcal{C}=\{000, 011, 101, 110\}$, specifically for position 1, the APP can be found by applying the theorem of total probability using the structure of the code:

$$\Pr(\mathsf{X}_1=0|\mathbf{y}) = \Pr(\mathsf{X}_1\mathsf{X}_2\mathsf{X}_3 = 000 \text{ or } 011|\mathbf{y}) \tag{3.21}$$

$$\Pr(\mathsf{X}_1=1|\mathbf{y}) = \Pr(\mathsf{X}_1\mathsf{X}_2\mathsf{X}_3 = 101 \text{ or } 110|\mathbf{y}) \tag{3.22}$$

For probability domain decoding, the decoder input is $q_i(x) = \Pr(\mathsf{X}_i = x|\mathsf{Y} = y_j)$ for $x = 0, 1$ and $i = 1, 2, \ldots, n$ and we want to find the APP output $r_i(x) = \Pr(\mathsf{X}_i|\mathbf{Y})$ (here, the roles of r and q as input and output have switched). Continuing the $n=3$ example, the APP value $r_1(x)$ using the inputs $q_1(x)$, $q_2(x)$ and $q_3(x)$:

$$r_1(0) = \Pr(\mathsf{X}_1=0|\mathbf{y}) = q_1(0)q_2(0)q_3(0) + q_1(0)q_2(1)q_3(1) \tag{3.23}$$

$$r_1(1) = \Pr(\mathsf{X}_1=1|\mathbf{y}) = q_1(1)q_2(0)q_3(1) + q_1(1)q_2(1)q_3(0) \tag{3.24}$$

because the channel is memoryless. In (3.23), each term has codewords terms with $x_1=0$, that is $(0,0,0)$ and $(0,1,1)$. Similarly, in (3.24), each term has codewords terms with $x_1=1$, that is $(1,0,1)$ and $(1,1,0)$. For general n, the APP decoder output $r_i(x)$ for position i is:

$$r_i(0) = \sum_{\substack{x_j : \sum x_j = 0 \\ x_i = 0}} \prod_{k=1}^{n} q_k(x_k) \tag{3.25}$$

$$r_i(1) = \sum_{\substack{x_j : \sum x_j = 0 \\ x_i = 1}} \prod_{k=1}^{n} q_k(x_k) \tag{3.26}$$

The restriction $x_j : \sum x_j = 0$ means to take the sum over only those code bits that sum to 0. Note that $r_i(0) + r_i(1) = 1$, so it is not necessary to compute both terms.

The MAP decision, or hard decision, for bit position i is made using the APP values:

$$\widehat{x}_i = \begin{cases} 0 & \text{if } r_i(0) \geq 0.5 \\ 1 & \text{if } r_i(0) < 0.5 \end{cases} \tag{3.27}$$

Example 3.2. A $(3, 2)$ single-parity check code is transmitted over the binary symmetric channel with error probability 0.2:

$$p_{\mathsf{Y}|\mathsf{X}}(y|x) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix} \tag{3.28}$$

Find the APP and MAP estimates when $\mathbf{y} = [010]$.

First, apply Bayes rule and note that in any of the three positions, if $y = 0$, then $q(0) = 0.8, q(1) = 0.2$. On the other hand, if $y = 1$ then $q(0) = 0.2, q(1) = 0.8$. Then the APP estimates are:

$$\begin{aligned} r_1(0) &= q_1(0)q_2(0)q_3(0) + q_1(0)q_2(1)q_3(1) \\ &= 0.8 \cdot 0.2 \cdot 0.8 + 0.8 \cdot 0.8 \cdot 0.2 \\ &= 0.256 \\ r_2(0) &= q_1(0)q_2(0)q_3(0) + q_1(1)q_2(0)q_3(1) \\ &= 0.8 \cdot 0.2 \cdot 0.8 + 0.2 \cdot 0.2 \cdot 0.2 \\ &= 0.136 \\ r_3(0) &= q_1(0)q_2(0)q_3(0) + q_1(1)q_2(1)q_3(0) \\ &= 0.8 \cdot 0.2 \cdot 0.8 + 0.2 \cdot 0.8 \cdot 0.8 \\ &= 0.256 \end{aligned}$$

The MAP decisions are:

$$(\widehat{x}_1, \widehat{x}_2, \widehat{x}_3) = (0, 0, 0) \tag{3.29}$$

In this case, the MAP decision is a codeword. In general, the MAP decision is not guaranteed to be a codeword.

3.1.5 Maximum Likelihood Decoding on AWGN Channel

ML decoding is an optimal decoding metric. It selects the codeword that maximizes the likelihood $\Pr(\mathbf{y}|\mathbf{c})$, that is the output is $\widehat{\mathbf{c}}$:

$$\widehat{\mathbf{c}} = \arg \max_{\mathbf{c} \in \mathcal{C}} \Pr(\mathbf{y}|\mathbf{c}) \tag{3.30}$$

One way to perform maximum likelihood decoding is to compare each codeword to the received sequence, and select the codeword for which $\Pr(\mathbf{y}|\mathbf{c})$ is greatest.

Consider specifically transmitting a binary codeword $\mathbf{c}$ over the AWGN channel. Binary code symbols $c \in \{0, 1\}$ are mapped to $x \in \{+1, -1\}$, according to $0 \to +1$ and $1 \to -1$. In what follows, convert from x to c using $x = 1 - 2c$.

Algorithm 3.1 Maximum Likelihood Decoding on AWGN Channel

Require: Received sequence $\mathbf{y}$. Generator matrix $\mathbf{G}$.

Ensure: Estimated codeword $\widehat{\mathbf{c}}$.

$M_{\text{best}} \leftarrow \infty$
Decode:
for $i = 0, 1, \ldots, 2^k - 1$ **do**
  $\mathbf{c}_i = \mathbf{uG}$ where $\mathbf{u}$ is binary representation of i
  $\mathbf{x} = 1 - 2\mathbf{c}_i$
  $M = \sum_{i=1}^{n} (x_i - y_i)^2$
  if $M < M_{\text{best}}$ **then**
    $\widehat{\mathbf{c}} \leftarrow \mathbf{c}_i$
    $M_{\text{best}} \leftarrow M$
  end if
end for

Because the AWGN channel is memoryless:

$$\Pr(\mathbf{y}|\mathbf{c}) = \prod_{i=1}^{n} \frac{1}{2\pi\sigma^2} e^{-\frac{(x_i - y_i)^2}{2\sigma^2}} \tag{3.31}$$

It is convenient to instead maximize $\ln \Pr(\mathbf{y}|\mathbf{c})$:

$$\ln \Pr(\mathbf{y}|\mathbf{c}) = \ln \prod_{i=1}^{n} \frac{1}{2\pi\sigma^2} e^{-\frac{(x_i - y_i)^2}{2\sigma^2}} \tag{3.32}$$

$$= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - y_i)^2 + n \ln 2\pi\sigma^2 \tag{3.33}$$

Because we are looking for the arg max, we can ignore the terms $1/(2\sigma^2)$ and $n \ln 2\pi\sigma^2$. Thus, the maximum likelihood solution is:

$$\widehat{\mathbf{c}} = \arg\min_{\mathbf{c} \in \mathcal{C}} ||\mathbf{x} - \mathbf{y}||^2, \tag{3.34}$$

where

$$||\mathbf{x} - \mathbf{y}||^2 = \sum_{i=1}^{n} (x_i - y_i)^2. \tag{3.35}$$

Again, take care that we convert from c to x using $x = 1 - 2c$. In other words, we find the codeword which is closest in the Euclidean distance sense.

Maximum likelihood decoding for the binary-input AWGN channel is implemented in Algorithm 3.1. However, since there are 2^k codewords, the complexity is exponential in the number of information symbols k. This is only practical codes where k is small.

3.2 Erasure Decoding

In an erasure channel, a transmitted symbol is either received correctly, or erased with a symbol ?. Using a binary code, assume transmission on an erasure channel: that is, 0 or 1 is changed to ? with some probability, but if a 0 or 1 is received, it is always correct. An error-correcting code can correct erasures:

Proposition 3.1. An error-correcting code $\mathcal{C}$ with minimum distance $d_{\min}$ can correct any pattern of s erasures if:

$$s \leq d_{\min} - 1 \tag{3.36}$$

Let $\mathbf{x}$ be the transmitted codeword and let $\mathbf{y}$ be the received sequence with erasures. If $\mathbf{x}'$ is any other codeword, then $\mathbf{x}$ and $\mathbf{x}'$ differ in at least $d_{\min}$ positions, but the erasures hide at most $d_{\min} - 1$ positions. Thus there is at least one non-erased position where $\mathbf{x}'$ and $\mathbf{x}$ differ. Since $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}'$ is not consistent with $\mathbf{y}$ in at least one position, the decoder can correctly choose $\mathbf{x}$.

This is easy to see with an example. Assume the codebook consists of $\mathcal{C} = \{0000000, 0110011\}$, and it has $d_{\min} = 4$. If the received sequence has 3 erasures,

$$\mathbf{y} = [0??00?1]^{\mathrm{t}} \tag{3.37}$$

then decoding is successful – clearly $\widehat{\mathbf{c}} = [0110011]^{\mathrm{t}}$. But if there are 4 erasures $\mathbf{y} = [0??00??]^{\mathrm{t}}$, then it is impossible to distinguish the two codewords.

As suggested by the proof of Proposition 3.1, erasure decoding can be regarded as solving a system of equations. If $\mathbf{y}$ is the received sequence, and an erasure "?" in position i is replaced with an unknown code bit x_i, then a method of decoding is to solve:

$$\mathbf{H}\mathbf{y}^{\mathrm{t}} = \mathbf{0}, \tag{3.38}$$

if possible.

Example 3.3. For a code with parity check matrix,

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

suppose that a codeword $\mathbf{x}$

$$\mathbf{x} = \big[0, 1, 1, 1, 0, 1\big]$$

is transmitted and the sequence

$$\mathbf{y} = \big[?, 1, 1, 1, ?, ?\big]$$

is received. Perform erasure decoding.

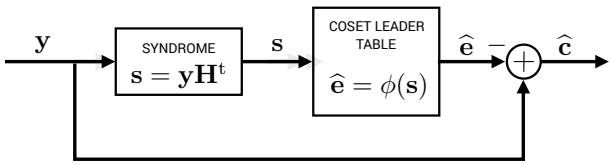


Figure 3.2: Syndrome decoder. The decoder input from the channel is $\mathbf{y}$. The syndrome $\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{t}}$ is computed, the estimated error $\widehat{\mathbf{e}} = \phi(\mathbf{s})$ is found from the coset leader table.

Replace the erasures with variables x_i, so that

$$\mathbf{y} = \big[x_1, 1, 1, 1, x_5, x_6\big]$$

and then the system of equations $\mathbf{H}\mathbf{y} = 0$ is:

$$\begin{aligned} x_1 + 1 + 1 &= 0 \\ 1 + x_5 + x_6 &= 0 \\ x_1 + 1 + x_6 &= 0 \\ 1 + 1 + x_5 &= 0 \end{aligned}$$

The solution is: $x_1 = 0$ and $x_5 = 0$, and then $x_6 = 1$, and the decoded sequence is:

$$\widehat{\mathbf{c}} = \big[x_1, 1, 1, 1, x_5, x_6\big] = \big[0, 1, 1, 1, 0, 1\big]$$

which is the transmitted codeword.

3.3 Syndrome Decoding

3.3.1 Decoding Cosets and Syndromes

Syndrome decoding is a hard-input method for decoding linear block codes. It works best when $n - k$ is small. For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a codeword $\mathbf{c} \in \mathcal{C}$ is transmitted over a channel that adds an error vector $\mathbf{e}$:

$$\mathbf{e} = [e_1\ e_2\ e_3\ \ldots\ e_n], \tag{3.39}$$

to the codeword, with $e_i \in \mathbb{F}$. The channel output is $\mathbf{y} = \mathbf{c} + \mathbf{e}$, and addition is in the field $\mathbb{F}_q^n$. For binary codes in $\mathbb{F}_2$, this corresponds to the binary symmetric channel (BSC).

Recall from Section **??** that if $\mathcal{G}$ is a group and $\mathcal{H}$ is a subgroup, then there exists a quotient group $\mathcal{G}/\mathcal{H}$. This partitions $\mathcal{G}$ into non-overlapping subsets called cosets, for example:

$$a + \mathcal{H} = \{a + h \mid h \in \mathcal{H}\} \tag{3.40}$$

is the coset containing a. From each coset, we can select one element to be the coset leader. For linear block codes, the group is the whole space $\mathbb{F}_q^n$ and the subgroup is the code $\mathcal{C}$.

Suppose that $\mathbf{c}$ is transmitted over the channel, noise $\mathbf{e}$ is added, so that the received symbol is $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where addition is in $\mathbb{F}_q$. From this, it can be seen that $\mathcal{C} + \mathbf{e}$ is the coset containing $\mathbf{e}$. The coset containing $\mathbf{a} = 0$ is the code itself. For any other $\mathbf{a} \neq 0$, no element of the coset containing $\mathbf{a}$ is a codeword.

The decoder input is $\mathbf{y} = \mathbf{x} + \mathbf{e}$. If the decoder can identify which coset $\mathbf{y}$ belongs to, then it can find a candidate for $\mathbf{e}$. We choose the coset leaders to be low-weight error vectors, since these are most likely. To identify the coset, we can compute a vector called the syndrome.

Definition 3.1. For a code $\mathcal{C}$ with $m \times n$ parity check matrix $\mathbf{H}$, the *syndrome* $\mathbf{s}$ of a sequence $\mathbf{y}$ is:

$$\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{t}}, \tag{3.41}$$

where $\mathbf{s}$ is a $1 \times m$ vector and $\mathbf{y}$ is a $1 \times n$ vector.

The syndrome of a codeword is $\mathbf{0}$. That is, $\mathbf{c} \in \mathcal{C}$ if and only if $\mathbf{c}\mathbf{H}^{\mathrm{t}} = \mathbf{0}$.

Example 3.4. Find the syndrome of $\mathbf{y}_1 = (0, 0, 0, 1, 1)$ and $\mathbf{y}_2 = (1, 0, 1, 1, 0)$ for the parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.42}$$

The syndromes are $\mathbf{s}_1 = \mathbf{y}_1\mathbf{H}^{\mathrm{t}} = (0, 1, 1)$ and $\mathbf{s}_2 = \mathbf{y}_2\mathbf{H}^{\mathrm{t}} = (0, 0, 0)$. Here, $\mathbf{y}_2$ is a codeword, but $\mathbf{y}_1$ is not.

Two sequences in the same coset can be described using the same error vector. For two codewords $\mathbf{c}', \mathbf{c}''$, let $\mathbf{e}$ be some fixed error vector, and the received sequences $\mathbf{y}'$ and $\mathbf{y}''$ are:

$$\mathbf{y}' = \mathbf{c}' + \mathbf{e} \text{ and} \tag{3.43}$$
$$\mathbf{y}'' = \mathbf{c}'' + \mathbf{e}. \tag{3.44}$$

Then $\mathbf{y}'$ and $\mathbf{y}''$ are in the same coset, by the definition of a coset. There is a one-to-one correspondence between cosets and syndromes:

Proposition 3.2. Two sequences are in the same coset if and only if they have the same syndrome.

Proof Compute the syndrome $\mathbf{s}'$ of $\mathbf{y}'$ and the syndrome $\mathbf{s}''$ of $\mathbf{y}''$:

$$\mathbf{s}' = \mathbf{y}'\,\mathbf{H}^{\mathrm{t}} = (\mathbf{c}' + \mathbf{e})\mathbf{H}^{\mathrm{t}} = \mathbf{c}'\,\mathbf{H}^{\mathrm{t}} + \mathbf{e}\mathbf{H}^{\mathrm{t}} = \mathbf{e}\mathbf{H}^{\mathrm{t}} \text{ and} \tag{3.45}$$
$$\mathbf{s}'' = \mathbf{y}''\mathbf{H}^{\mathrm{t}} = (\mathbf{c}'' + \mathbf{e})\mathbf{H}^{\mathrm{t}} = \mathbf{c}''\mathbf{H}^{\mathrm{t}} + \mathbf{e}\mathbf{H}^{\mathrm{t}} = \mathbf{e}\mathbf{H}^{\mathrm{t}} \tag{3.46}$$

since $\mathbf{c}\mathbf{H}^{\mathrm{t}} = \mathbf{0}$ for all codewords $\mathbf{c}$. That is, $\mathbf{y}'$ and $\mathbf{y}''$ have the same syndrome and are in the same coset. The syndrome is independent of the codeword. □.

A coset has many members, we will identify the coset by one of its members — in particular, a member with the lowest weight:

Definition 3.2. The one coset word of minimum weight is called the *coset leader*. In the case of a tie, one coset word is arbitrarily designated the coset leader.

The *standard array* for a binary linear code $\mathcal{C}$ is a table with 2^{n-k} rows and 2^k columns such that:

- its 2^n entries are all 2^n vectors in $\mathbb{F}_2^n$,
- each row contains a distinct coset of $\mathcal{C}$,
- the first column is the coset leaders,
- the first row is the codebook $\mathcal{C}$.

Example 3.5. Find the standard array for the binary code $\mathcal{C}$ with generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{3.47}$$

The codebook is $\mathcal{C} = \{00000, 10110, 01110, 11101\}$ and the parity check matrix is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.48}$$

The standard array is given in Fig. 3.3.

The second column shows errors with respect to the all-zeros codeword. These are the correctable errors vectors, which includes all weight-one sequences. All of the sequences in the second row consist of the correctable error vectors. If the sequence (1 0 1 1 1) is received, the nearest codeword is (1 0 1 1 0), corresponding to an error $\mathbf{e} = (0\ 0\ 0\ 0\ 1)$.

Note that in each of the last two rows of the standard array, there are two sequences of weight two. The choice of (00101) and (01100) was made arbitrarily.

3.3.2 Syndrome Decoding

The goal of syndrome decoding is to estimate the error vector. This error vector is then subtracted from the decoder input to obtain the codeword estimate. Since all elements of a coset have the same syndrome, and they have the same error vector. In syndrome decoding, the syndrome is found; using the syndrome,

syndrome **s**	Standard Array				
0 0 0	0 0 0 0 0	1 0 1 1 0	0 1 0 1 1	1 1 1 0 1	codebook $\mathcal{C}$
0 0 1	0 0 0 0 1	1 0 1 1 1	0 1 0 1 0	1 1 1 0 0	a coset
0 1 0	0 0 0 1 0	1 0 1 0 0	0 1 0 0 1	1 1 1 1 1	
0 1 1	0 1 0 0 0	1 1 1 1 0	0 0 0 1 1	1 0 1 0 1	
1 0 0	0 0 1 0 0	1 0 0 1 0	0 1 1 1 1	1 1 0 0 1	
1 1 0	1 0 0 0 0	0 0 1 1 0	1 1 0 1 1	0 1 1 0 1	
1 0 1	0 0 1 0 1	1 0 0 1 1	0 1 1 1 0	1 1 0 0 0	
1 1 1	0 1 1 0 0	1 1 0 1 0	0 0 1 1 1	1 0 0 0 1	
	coset leaders		decoding region around 01011		

Figure 3.3: Standard array for Example 3.5.

the error vector is identified, and then the error vector is subtracted from the received sequence.

Given the syndrome, the coset leader is interpreted as the *error vector* or *error pattern*, since the coset leader has the lowest weight. Syndrome decoding is described in Algorithm 3.2.

A table $\phi(\mathbf{s}) = \widehat{\mathbf{e}}$ provides the coset leader $\widehat{\mathbf{e}}$ from $\mathbf{s}$. The syndrome table for Example 3.5 is given in Table 3.1. The syndrome table may be generated by finding the syndromes for error patterns of increasing weight. Clearly, the error pattern $\widehat{\mathbf{e}} = \mathbf{0}$ corresponds to $\mathbf{s} = \mathbf{0}$. Then, consider all sequences of weight one, for example $\widehat{\mathbf{e}} = 00001$ corresponds to $\widehat{\mathbf{e}}\mathbf{H}^{\mathrm{t}} = 001$, so $\phi(001) = 000001$. The syndrome table also contains error patterns of weight 2, but a given syndrome may have multiple error patterns of the same weight; in this case, choose one error pattern arbitrarily.

SSQ 3.1. For the (7,4) Hamming code with parity check matrix $\mathbf{H}$:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

there are 8 syndromes. For each syndrome, find the corresponding error pattern.

Example 3.6. Continuing Example 3.5, decode the received sequence $\mathbf{y} =$

Algorithm 3.2 Syndrome Decoding

Require: Received sequence $\mathbf{y}$. Parity-check matrix $\mathbf{H}$ and syndrome table ϕ.

Ensure: Estimated codeword $\widehat{\mathbf{c}}$.

Compute the syndrome $\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{t}}$

Find $\widehat{\mathbf{e}} = \phi(\mathbf{s})$

Output the estimated codeword is $\widehat{\mathbf{c}} = \mathbf{y} - \widehat{\mathbf{e}}$.

Table 3.1: Syndrome table for Example 3.5

syndrome $\mathbf{s}$	coset leader $\widehat{\mathbf{e}} = \phi(\mathbf{s})$
0 0 0	0 0 0 0 0
1 0 0	0 0 1 0 0
0 1 0	0 0 0 1 0
1 1 0	1 0 0 0 0
0 0 1	0 0 0 0 1
1 0 1	0 0 1 0 1
0 1 1	0 1 0 0 0
1 1 1	0 1 1 0 0

(0 0 1 1 0) using syndrome decoding. The syndrome is $\mathbf{s} = \mathbf{y}\mathbf{H}^t = (1\ 1\ 0)$. Referring to the table, the corresponding error vector is $\widehat{\mathbf{e}} = (1\ 0\ 0\ 0\ 0)$. Then, the estimated codeword is $\widehat{\mathbf{c}} = \mathbf{y} - \widehat{\mathbf{e}} = (1\ 0\ 1\ 1\ 0)$.

3.4 Performance of Error-Correcting Codes

3.4.1 Word error rate and bit error rate

Word-error rate (WER) and bit-error rate (BER) are two measures of a decoders performance. Recall that a codeword $\mathbf{c}$ is transmitted over a channel, the sequence $\mathbf{y}$ is received, and the decoder estimates the transmitted sequence as $\widehat{\mathbf{c}}$. A codeword $\mathbf{c}$ is transmitted, $\mathbf{y}$ is received, and the decoder possibly outputs $\widehat{\mathbf{c}}$. There are three possible decoding results:

Decoder succeeds The output is correct, that is $\widehat{\mathbf{c}} = \mathbf{c}$.

Decoder error The output is incorrect, that is $\widehat{\mathbf{c}} \neq \mathbf{c}$.

Failure to decode The decoder reports that it cannot find a codeword. Certain decoding algorithms recognize that the input sequence cannot be decoded. Denote this as an event E.

The probability of decoding error is an important metric to measure both the goodness of the code $\mathcal{C}$ and its error-correcting algorithm.

The *probability of word error* or word error rate (WER) is the probability that the decoded word $\widehat{\mathbf{c}}$ is not equal to the transmitted word $\mathbf{c}$, that is:

$$\text{WER} = \Pr[\mathbf{c} \neq \widehat{\mathbf{c}}], \tag{3.49}$$

The probability of bit error or *bit error rate* (BER) is the probability that the decoded code bit $\widehat{c}_i$ is not equal to the transmitted code bit c_i:

$$\text{BER} = \Pr[\widehat{c}_i \neq c]. \tag{3.50}$$

When determining WER and BER, the event E is regarded as an error. Usually WER and BER are obtained by Monte Carlo simulations, this is described in a following subsection

3.4.2 Evaluating Codes and Decoders Using Monte Carlo Simulations

The performance of an error-correcting code and its decoder is evaluated using WER and BER, as defined in Section 3.4.1. Monte Carlo simulations are most commonly used to evaluate the decoder error rate. The computer simulation models the encoder, the channel and the decoder, that is, the three parts of the communication system shown in Fig. 1.1. For a given channel, the WER and BER depends both on the code and the decoder.

The goal is to express WER and BER as a function of the channel parameter. As the channel gets "better", the WER and BER decrease. Consider the example of the code in Example 3.5, transmitted over a binary symmetric channel (BSC) with error probability p, and is decoded using syndrome decoding. The WER is shown in Fig. 3.4. The "better" channel has smaller p, and the WER for the decoder decreases as p decreases.

A "word error" means even one bit is in error. For example, if

$$\mathbf{u} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.51}$$

is the transmitted information and

$$\widehat{\mathbf{u}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.52}$$

is the decoded information sequence, then this represents one word error and three bit errors.

3.4.3 Source Code for Syndrome Decoding

The following simple example illustrates the principles of a computer simulation. Source code is provided using Matlab, but the principles apply for any programming language. The following example uses the code in Example 3.5, transmitted over a binary symmetric channel (BSC) with error probability p, and is decoded using syndrome decoding.

The first step is to generate $k = 2$ equally likely bits of information $\mathbf{u}$, from the set $\{0, 1\}$. Matlab has command which does this:

```
k = 2;
u = randi([0 1],1,k);
```

Encoding may performed using the generator matrix:

```
c = u * G;
```

So that the codeword $\mathbf{c}$ is a 1-by-5 vector.

Channel noise must be simulated. A binary symmetric channel with error probability p can be simulated by generating an error vector $\mathbf{z}$ which has a 1 with probability p. The vector $\mathbf{z}$ is added to $\mathbf{c}$ to obtain the channel output $\mathbf{y}$:

```
p = 0.03;
z = rand(1,n) < p;
y = mod(c + z,2)
```

The decoder is usually the most complicated part of the simulation, but for syndrome decoding is only three lines of source code. The input is the channel sequence $\mathbf{y}$, and the output is the estimated codeword $\widehat{\mathbf{c}}$ or the estimated information $\widehat{\mathbf{u}}$. This is an implementation of the syndrome decoder:

```
s = mod(y * H',2);
ehat = Phi(bi2de(s)+1,:);
chat = mod(y - ehat,2);
```

Finally, the number of bit errors and word errors are counted. The number of bit errors is the number of positions where $\mathbf{c}$ and $\widehat{\mathbf{c}}$ disagree. The number of word errors is either 0 or 1. If the number of bit errors is greater than 0, then a word error occurred:

```
numberOfBitErrors = sum( mod( c - chat, 2) );
numberOfWordErrors = (numberOfBitErrors > 0);
```

Matlab source code for the simulation is shown in Fig. 3.5.

3.4.4 Estimating Error Rates

In order to estimate the WER and BER, Monte Carlo simulations are performed.

That is, the above simulation is repeated for many frames.

After accumulating errors over many frames, the WER can be estimated as:

$$\text{WER} \approx \frac{\text{number of word errors}}{\text{total number of frames}} \tag{3.53}$$

and the BER can be estimated as:

$$\text{BER} \approx \frac{\text{number of bit errors}}{\text{block length} \times \text{total number of frames}} \tag{3.54}$$

More precisely, let N_{word} be the number of word errors and let N be the number of frames. Then,

$$\text{WER} \approx \frac{N_{\text{word}}}{N} \tag{3.55}$$

Let N_{bit} be the number of bit errors. Then,

$$\text{BER} \approx \frac{N_{\text{bit}}}{n \cdot N} \tag{3.56}$$

where n is the code block length.

Each frame is an experiment which either "passes" (decoding succeeds) or "fails" (decoding fails); the probability of failure is WER. Each frame is an independent experiment, and we repeat the experiment many times to obtain an estimate of WER.

How many frames should be simulated? That is, how large should N be? If N is small, we can get a result with a small amount of computer time, but the WER estimate will not be very reliable. In the following example, $N = 7000$ is used. For each of 5 simulations, the computer time is low, but the variation in WER is large, from 0.0001428 to 0.001142.

```
>> syndromeDecodingExample(0.008,7000)
Nerr = 6, WER = 0.0008571, computer time = 0.11 seconds
>> syndromeDecodingExample(0.008,7000)
Nerr = 3, WER = 0.0004285, computer time = 0.11 seconds
>> syndromeDecodingExample(0.008,7000)
Nerr = 2, WER = 0.0002857, computer time = 0.12 seconds
>> syndromeDecodingExample(0.008,7000)
Nerr = 8, WER = 0.001142, computer time = 0.11 seconds
>> syndromeDecodingExample(0.008,7000)
Nerr = 1, WER = 0.0001428, computer time = 0.12 seconds
```

Large variation leads to inconsistent results, and it is better to choose N large enough so that we can have a reliable WER estimate after one simulation. In the following example, N increases to 700000. For each of 5 simulations, the WER variation is smaller, from 0.0004685 to 0.0005328, which indicates a better WER estimate. However, more computer time is required:

```
>> syndromeDecodingExample(0.008,700000)
Nerr = 373, WER = 0.0005328, computer time = 9.57 seconds
>> syndromeDecodingExample(0.008,700000)
Nerr = 337, WER = 0.0004814, computer time = 9.5 seconds
>> syndromeDecodingExample(0.008,700000)
Nerr = 334, WER = 0.0004771, computer time = 9.48 seconds
>> syndromeDecodingExample(0.008,700000)
Nerr = 367, WER = 0.0005242, computer time = 9.45 seconds
>> syndromeDecodingExample(0.008,700000)
Nerr = 361, WER = 0.0005157, computer time = 9.43 seconds
>> syndromeDecodingExample(0.008,700000)
Nerr = 328, WER = 0.0004685, computer time = 9.43 seconds
```

Pro Tip When running a simulation, good channels with high SNR require more frames for accurate WER and BER estimates. This is in comparison to bad channels with a low SNR, which require fewer frames. A good approach is to stop the simulation when the number of word errors N_{word} reaches a fixed value, instead of stopping when the number of frames N reaches a fixed value. For example, repeat the simulation until 100 word errors have been accumulated, instead of repeating $100,000$ simulation. This approach gives good reliability for a fixed amount of computer time, whether the channel is good or bad. To implement this in Fig. 3.5, change the while statement to `while numberOfWordErrors < Nstop`. In practice, simulating 5 or 10 frame errors can be used for preliminary results, but publication-quality data should use 50 to 100 word errors for high reliability.

Using N_{word} as the stopping condition is to preferred to using N_{bit} because word errors are statistically independent events, but bit errors are not.

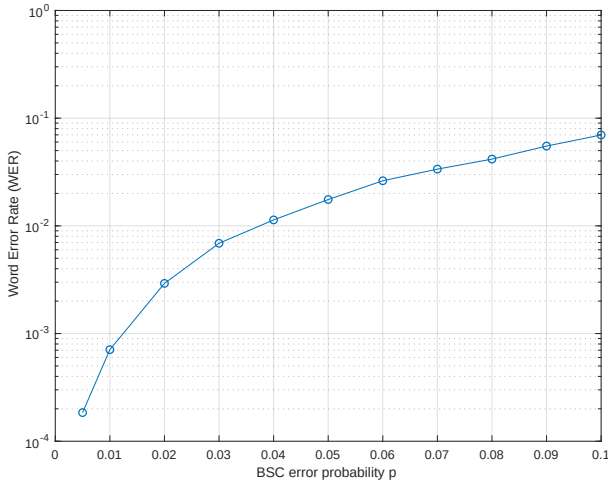


Figure 3.4: Word error rate (WER) for syndrome decoding of the code in Example 3.5 on the BSC with error probability p.

In summary, the value of N should be chosen large enough to obtain a WER with high reliability, but low enough to avoid unnecessary computer time.

3.4.5 Signal-to-Noise Ratio for Binary-Input AWGN Channels

Consider specifically transmitting a binary codeword $\mathbf{c}$ over the AWGN channel. The binary-input AWGN channel model has inputs $c_1, c_2, \ldots, c_n \in \{0, 1\}$ are mapped to $x \in \{+1, -1\}$, according to $0 \to +1$ and $1 \to -1$. At the transmitter side, convert from x to c using $x = 1 - 2c$.

Consider two definitions of signal-to-noise ratio:

- The energy per symbol is $E_s = 1$, since $+1$ and -1 both have power 1.

Thus:

$$\frac{E_s}{N_0} = \frac{1}{2\sigma^2} \tag{3.57}$$

- There are n code symbols but only k information symbols for a rate $R = k/n$ code. If we are interested in the power per information symbol E_b, then we write $E_b = E_s/R$. Thus:

$$\frac{E_b}{N_0} = \frac{1}{2R\sigma^2} \tag{3.58}$$

  E_b is the average power of transmitting one information symbol, which is a reasonable goal.

E_b/N_0 is usually expressed in dB, that is:

$$\frac{E_b}{N_0}(\text{dB}) = 10\log_{10}\frac{E_b}{N_0} \tag{3.59}$$

When you write a simulation, you typically fix E_b/N_0 (dB) for fixed values, for example 1 dB, 2 dB, 3 dB. Then, find σ^2 as:

```
EbNo = 10^(EbNodB / 10)
var = 1 / (2 * EbNo*R)
z = randn(1,n) * sqrt(var)
```

so that `z` is AWGN noise, and the channel output is:

```
y = x + z
```
(3.60)

as your channel output. If you are working with E_s/N_0, then replace `EbNo` with `EsNo` and remove `R`.

3.5 Exercises

3.1 Consider the code with parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.61}$$

Perform erasure decoding with the received sequences:

(a) $\mathbf{y}_1 = [0, 0, 1, ?, ?, ?]$
(b) $\mathbf{y}_2 = [?, ?, ?, 1, 0, 1]$
(c) $\mathbf{y}_3 = [?, ?, ?, 0, 1, ?]$

For each one, give the estimated codeword $\widehat{\mathbf{c}}$, or "failure to decode"

```
function syndromeDecodingExample(p,Nstop)

G = [...                    %Generator matrix
     1 0 1 1 0 ; ...
     0 1 0 1 1];
H = [...                    %Parity-check matrix
     1 0 1 0 0 ; ...
     1 1 0 1 0 ; ...
     0 1 0 0 1];
[k,n] = size(G);

Phi = [ ...                 %Syndrome decoder table
     0 0 0 0 0 ; ...
     0 0 1 0 0 ; ...
     0 0 0 1 0 ; ...
     1 0 0 0 0 ; ...
     0 0 0 0 1 ; ...
     0 0 1 0 1 ; ...
     0 1 0 0 0 ; ...
     0 1 1 0 0 ];

numberOfBitErrors  = 0;
numberOfWordErrors = 0;
Nframes = 0;

while Nframes < Nstop
    %encoder
    u = randi([0 1],1,k);
    c = mod(u * G,2);

    %channel
    e = rand(1,n) < p;
    y = mod(c+e,2);

    %syndrome decoding
    s = mod(y * H',2);
    ehat = Phi(bi2de(s)+1,:);  %binary-to-decimal
        conversion
    chat = mod(y - ehat,2);

    nbe                 = sum( mod( c - chat, 2) );
    numberOfBitErrors   = numberOfBitErrors + nbe;
    numberOfWordErrors = numberOfWordErrors + (nbe > 0);
    Nframes             = Nframes + 1;
end
WER = numberOfWordErrors / Nframes;
BER = numberOfBitErrors / (n * Nframes);
fprintf('Nerr = %d, WER = %G, BER = %G\n',
    numberOfWordErrors,WER,BER);
save decoding_data  %change filename to avoid overwrite
                    %another .m file loads and plots
```

Figure 3.5: Matlab source code implementing simulation using syndrome decoding.

3.2 By computer simulation, find the probability of word error of the $(7, 4)$ Hamming code with:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.62}$$

on the binary symmetric channel with probability of error p, using *syndrome decoding*. Find the generator matrix $\mathbf{G}$. In your simulation, randomly generate data $\mathbf{u}$, and find the corresponding codeword $\mathbf{c} = \mathbf{uG}$. Your syndrome decoder should output an estimated codeword $\widehat{\mathbf{c}}$. Use your simulation to estimate the probability of word error rate WER, for $p = 0.1, 0.025, 0.0025$

3.3 By computer simulation, find the BER of the (7,4) Hamming code given in Question 3.2 on the binary-input AWGN channel, using *maximum likelihood decoding*. Estimate the BER for each $E_s/N_0 = 1$ dB, 2 dB, ..., 9 dB. Make a plot of your results, with the BER on the vertical axis with a log scale, and the E_s/N_0 in dB on the horizontal axis. On the same graph, plot the bit-error rate for uncoded channel, found in Question 1.5.

Chapter 4

Low-Density Parity-Check Codes

Low-density parity-check codes have a sparse parity-check matrix. This enables efficient decoding algorithms which are proportional to the block length. Under certain conditions, the minimum distance increases with the block length. These two properties make LDPC codes very attractive.

4.1 Definition and Graphical Representation

4.1.1 Regular LDPC Codes

A low-density parity-check code is a code whose $m \times n$ parity check matrix $\mathbf{H}$ has a small number of ones. We also say the matrix is sparse or low-density.

Definition 4.1. For a *regular LDPC code*, the parity-check matrix $\mathbf{H}$ has d_{v} ones per column and d_{c} ones per row. We also say the column weight is d_{v} and the row weight is d_{c}.

Example 4.1. The following matrix has $d_{\mathsf{v}} = 2$ and $d_{\mathsf{c}} = 4$, and is a parity-check matrix for a block length $n = 8$ code:

$$\begin{bmatrix} 1 & \mathbf{1} & \mathbf{1} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

The above matrix does not appear sparse, it has equal numbers of zeros and ones. But as n increases, the sparse structure becomes more clear. The following matrix also has $d_{\mathsf{v}} = 2, d_{\mathsf{c}} = 4$, and is a parity-check matrix for block length

$n = 20$ code:

$$\begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

This matrix can be more clearly seen to be sparse.

For LDPC codes, a "small number of ones" means the number of ones in $\mathbf{H}$ grows linearly with the block length n. That is, if $\mathbf{H}$ has n columns, the total number of ones in $\mathbf{H}$ is nd_{v} ones. If $\mathbf{H}$ has m columns, then the total number of ones is md_{c}. So, while the number of ones grows linearly in n, the total number of elements grows quadratically in n. This chapter concentrates on binary LDPC codes, but nonbinary LDPC codes are constructed over a q-ary field $\mathbb{F}_q$, and the density of non-zero elements is low.

The dimension and rate of an LDPC code depends on the rank of $\mathbf{H}$. For a general $m \times n$ matrix $\mathbf{H}$ with $m \leq n$, the rank of $\mathbf{H}$ is less than or equal to m:

$$\text{rank}(\mathbf{H}) \leq m \tag{4.1}$$

For regular LDPC code matrix, there may be linearly dependent rows in $\mathbf{H}$, so the rank of $\mathbf{H}$ is strictly less than m. This means that the dimension of the code is $k = n - \text{rank}(\mathbf{H})$ (not $n - m$) and the code rate is $R = 1 - \frac{\text{rank}(\mathbf{H})}{n}$. The design rate is defined using the size of the parity check matrix, without regard of the rank.

Definition 4.2. The *design rate* of a regular LDPC code with an $m \times n$ parity check matrix is:

$$R_{\text{d}} = 1 - \frac{m}{n}. \tag{4.2}$$

The design rate satisfies $R_{\text{d}} \leq R$. The design rate can be related to d_{v} and d_{c} because there are two ways to compute the number of ones in $\mathbf{H}$, that is $d_{\mathsf{c}}m = d_{\mathsf{v}}n$, that is $R_{\text{d}} = 1 - \frac{d_{\mathsf{v}}}{d_{\mathsf{c}}}$.

4.1.2 Tanner Graph

LDPC codes have a graph representation, which is useful when discussing LDPC codes. A simple undirected graph G consists of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$, which connect the vertices.

Definition 4.3. A simple undirected graph $G = (\mathcal{V}, \mathcal{E})$ is called a *bipartite graph* if there exists a partition of $\mathcal{V}$ so that both $\mathcal{V}_1$ and $\mathcal{V}_2$ are independent sets. Write $G = (\mathcal{V}_1 + \mathcal{V}_2, \mathcal{E})$ to denote a bipartite graph with partitions $\mathcal{V}_1$ and $\mathcal{V}_2$.

A *Tanner graph* is a bipartite graph corresponding to the $m \times n$ parity-check matrix $\mathbf{H}$. The Tanner graph has: n variable nodes (or bit nodes), represented by circles, m check nodes, represented by squares. There is an edge between variable node i and check node j if there is a one in row i and column j of $\mathbf{H}$. The Tanner graph corresponding to the check matrix in Example 4.1 is shown in Fig. 4.1.

Recall from Subsection 2.2.3 that one row of $\mathbf{H}$ is a parity check $\mathbf{h}$, which corresponds to a parity-check equation $\mathbf{hc}^{\mathrm{t}} = 0$. In the Tanner graph, the row, and thus the parity check constraint, is represented by a check node. For example, the first square in Fig. 4.1 is connected to x_1, x_2, x_3 and x_4, which indicates $x_1 + x_2 + x_3 + x_4 = 0$.

As can be seen from the Tanner graph, each variable node i is connected to several check nodes. Let $\mathcal{M}_i$ denote the set of check nodes that variable node i is connected to. Likewise, each check node j is connected to several variable nodes. Let $\mathcal{N}_j$ denote the set of variable nodes that check node j is connected to. This notation will be useful later on.

Example 4.2. The Tanner graph corresponding to Example 4.1 is shown in Fig. 4.1. In addition, the variable node and check node connectivity sets are given by:

$$\begin{array}{lll} \mathcal{N}_1 = \{1,2,3,4\} & \mathcal{M}_1 = \{1,3\} & \mathcal{M}_5 = \{2,4\} \\ \mathcal{N}_2 = \{5,6,7,8\} & \mathcal{M}_2 = \{1,4\} & \mathcal{M}_6 = \{2,3\} \\ \mathcal{N}_3 = \{1,4,6,8\} & \mathcal{M}_3 = \{1,4\} & \mathcal{M}_7 = \{2,4\} \\ \mathcal{N}_4 = \{2,3,5,7\} & \mathcal{M}_4 = \{1,3\} & \mathcal{M}_8 = \{2,3\} \end{array}$$

If a row of $\mathbf{H}$ has weight d, then the corresponding check node has degree d. Similarly, if a column of $\mathbf{H}$ has weight d, then the corresponding variable node has degree d. Irregular LDPC codes, where each row (and each column) may have varying weights, are described in Section 5.2.

Definition 4.4. A *cycle* of length L in a bipartite graph is a path of L edges which closes back on itself

The cycle length L in a bi-partite graph is even. The way we have defined a Tanner graph above, $L = 2$ cycles are not possible, so possible cycle lengths are $4, 6, 8, \ldots$.

Definition 4.5. The *girth* of a bipartite graph is the minimum cycle length of the graph.

Short cycles in a graph can reduce decoding performance. The girth is important because we want the length of the shortest cycle to be high, and further the number of such short cycles should be small. Graphs with high girth are desirable.

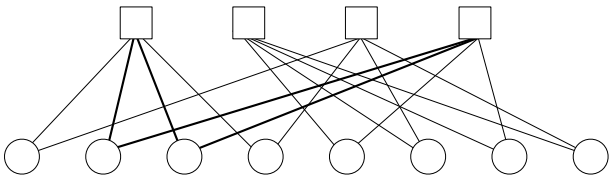


Figure 4.1: Tanner graph corresponding to the first matrix in Example 4.1.

4.2 Message-Passing Algorithms

4.2.1 Motivating Example for Message Passing

"Message passing decoding" is a broad class of algorithms which includes LDPC decoding algorithms. Before describing message-passing decoding of LDPC codes, we describe another problem which can be solved by message-passing method. It is a first taste of message algorithms. Also, this example is used to introduce an important concept, the sum-product update rule.

> Some soldiers are standing in a line. The Commander wants to count them. Calling out the name of each solider is too difficult. And, because there is fog, the commander cannot see very far. How to count the soldiers?

In this problem, each soldier can only communicate integers with his neighbors. Soldiers standing in a line is shown in Fig. 4.2-(a). The following message-passing algorithm allows the commander to obtain the count:

1. The soldiers at each end send a "1" message to their neighbors.
2. A soldier who receives a message, adds one to it, and sends it to his neighbor. If you get a message from the left, send it to the right
3. The commander receives messages from both the left and right, adds one for himself, to obtain the total number of soldier

Not only the commander, but all soldiers can find the total number.

The algorithm works even if the soldiers are standing in a "Y" as shown in Fig. 4.2-(b). The soldier with three neighbors receives two messages $r_1 = 2$ and $r_2 = 2$. This soldier then sends a message $q_3 = r_1 + r_2 = 4$ to the neighbor. Note however, that even though this soldier receives a message $r_3 = 3$, it is not used in computing the outgoing message r_3 on the same edge. This is an example of the sum-product update rule:

Definition 4.6. *Sum-Product Update Rule* For any node t, the outgoing message on edge e depends on the incoming messages on all edges connected to t except e.

However, for soldiers in a loop as shown in Fig. 4.2-(c), there is no simple message-passing solution.

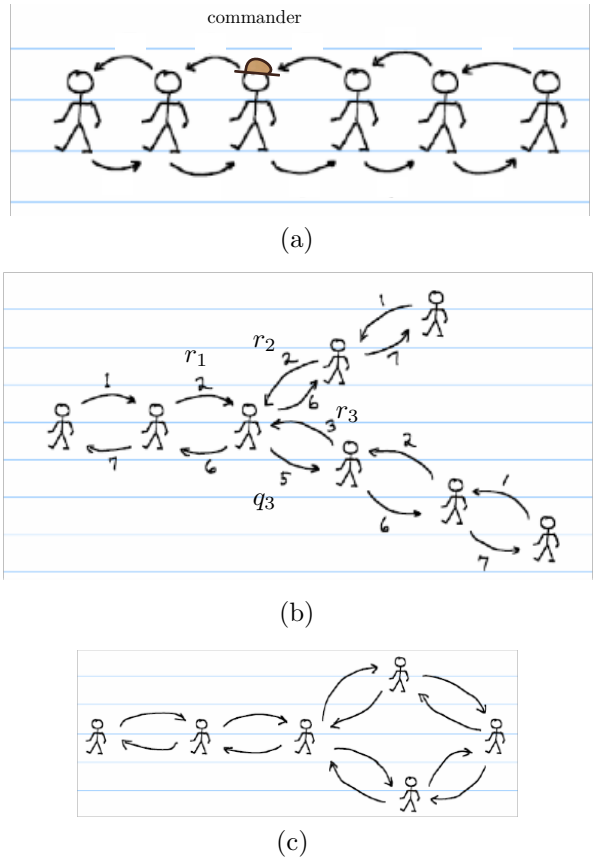


Figure 4.2: (a) Counting soldiers in a line. (b) Counting soldiers in a "Y." (c) There is no easy message-passing approach to counting soldiers in a loop. Figures due to William Ryan.

4.2.2 Message-Passing Decoding of LDPC Codes

The following steps express the general idea of LDPC decoding by applying the ideas of message-passing to the Tanner graph.

1. *Initalize* At each variable node, the message from the channel is sent to all connected check nodes

2. *Check Node Function* At each check node, compute each outgoing message using the other incoming messages. These messages are sent to the variable nodes.

3. *Variable Node Function* At each variable node, compute each outgoing message using the other incoming messages. These messages are sent to the check nodes.

4. *Hard Decisions* At each variable node i, make a hard estimate $\widehat{c}_i \in \{0, 1\}$

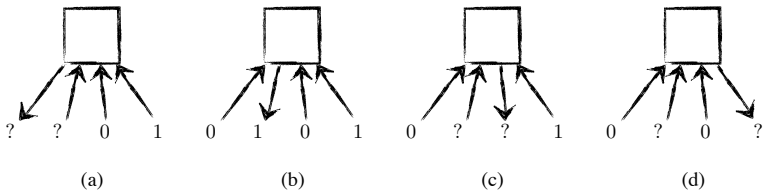


Figure 4.3: Decoding at a degree-4 check node with inputs $(q_1, q_2, q_3, q_4) = (0, ?, 0, 1)$. To compute an output r_i, use all the other inputs, and ignore the input q_i. The output is $(r_1, r_2, r_3, r_4) = (?, 1, ?, ?)$.

5. *Check for convergence*

   - If $\widehat{\mathbf{c}}\mathbf{H}^{\mathrm{t}} = 0$, then the decoder has found a valid codeword; output $\widehat{\mathbf{c}}$ and stop.
   - If $\widehat{\mathbf{c}}\mathbf{H}^{\mathrm{t}} \neq 0$ and maximum number of iterations reached, then output $\widehat{\mathbf{c}}$ and stop.
   - Otherwise, go to Step 2.

   There are several variations of LDPC decoding algorithms that follow these steps.

4.2.3 Erasure Decoding of LDPC Codes

LDPC decoding on the erasure channel is particularly simple, and is an example of message-passing algorithm. A codeword $\mathbf{c}$ from an LDPC code $\mathcal{C}$ with parity-check matrix $\mathbf{H}$ is transmitted over the erasure channel. Some of the 0's and 1's are replaced with an erasure symbol ?. Decoding is performed by applying message passing to the Tanner graph. The messages being passed are from the set $\{0, ?, 1\}$.

Check node function The check node finds each outgoing message as follows. If none of the incoming messages is ?, then the outgoing message satisfies the parity check equation. But, if any incoming message is ? then the outgoing message is ?. This follows the sum-product update rule.

For example, a degree 4 check node has $x_1 + x_2 + x_3 + x_4 = 0$. If the inputs are $(q_1, q_2, q_3) = (1, 0, 1)$ then the outgoing message is $r_4 = 0$. But if instead $(q_1, q_2, q_3) = (1, 0, ?)$, then the outgoing message is $r_4 = ?$. Another example is shown in Fig. 4.3, where the inputs are $(0, ?, 0, 1)$ and the outputs are $(?, 1, ?, ?)$.

Variable node function If at least one of the incoming messages is a 0 or a 1, then the outgoing message will be that message. But, if all of the incoming messages is ?, then the outgoing message is ?.

For example, a degree 4 variable node, if the input is $(y, r_1, r_2, r_3) = (?, ?, 0, ?)$ then the outgoing message is $q_4 = 0$. But if the input is $(y, r_1, r_2, r_3) = (?, ?, ?, ?)$ then the outgoing message is $q_4 = ?$. Another example is shown in Fig. 4.4, where the inputs are $(0, ?, 0, 1)$ and the outputs are $(?, 1, ?, ?)$. Note that specifically

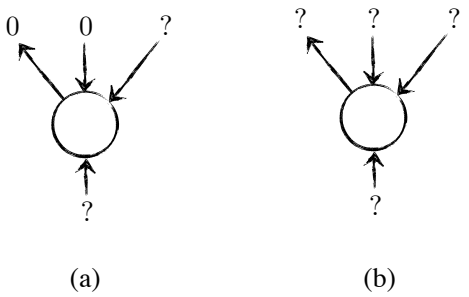


Figure 4.4: Decoding at a degree-3 variable node (a) With inputs $(r_2, r_3) = (0, ?)$ and $y =?$, the output is $q_1 = 0$. (b) With inputs $(r_2, r_3) = (?, ?)$ and $y =?$, the output is $q_1 =?$.

for the erasure channel, it will never occur that two incoming messages will be *both* a 0 and a 1, on two incoming messages at one variable node.

Hard decision The variable node makes a hard decision by including all messages, including the channel message. For example, if the input at the variable node is $(y, r_1, r_2, r_3) = (0, ?, ?, ?)$ then then hard decision output is $\widehat{x} = 0$. But, if $(r_1, r_2, r_3, r_4) = (?, ?, ?, ?)$ then $\widehat{x} =?$.

Example 4.3. A codeword from a code with parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

is transmitted over the erasure channel and the received sequence is:

$$\mathbf{y} = (?, 1, 1, 1, ?, ?) \tag{4.4}$$

The decoding process is illustrated in Fig. 4.5, and the final decoding result is:

$$\widehat{\mathbf{c}} = (0, 1, 1, 1, 0, 1)$$

4.3 Sum-Product Decoding

Sum-product is a soft-input decoding algorithm for LDPC codes. While not optimal, it is performs quite well in practice.

4.3.1 Algorithm

As with erasure decoding, sum-product decoding is message passing on a graph. The decoder has messages, which are passed along edges on graph, and the nodes

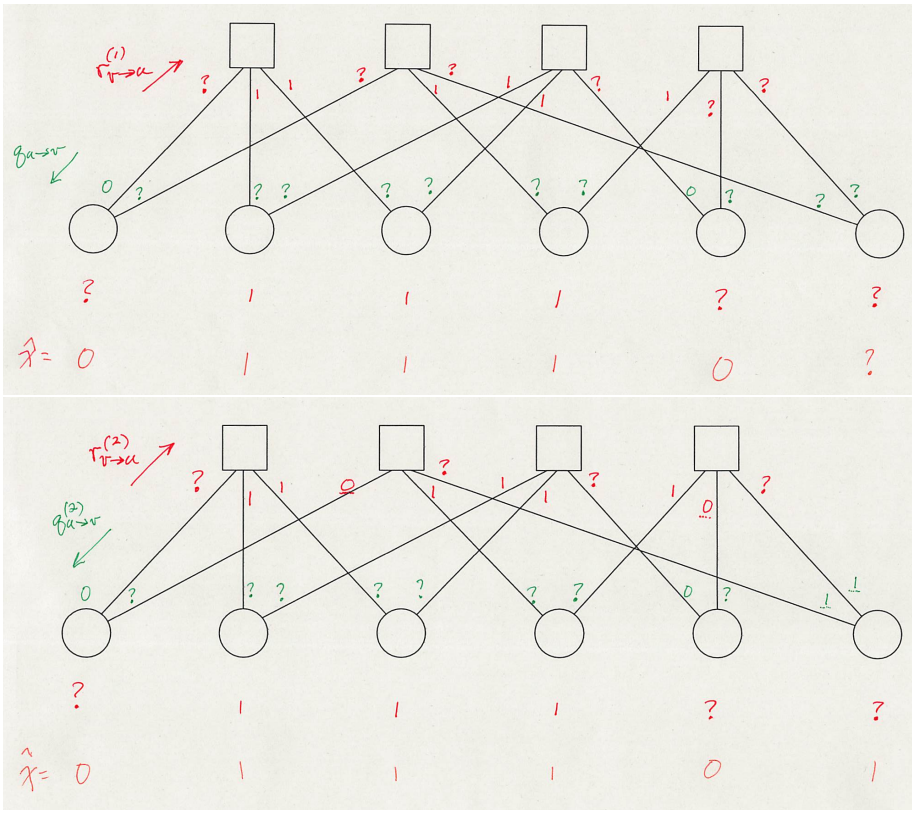


Figure 4.5: Example of decoding an LDPC code on the erasure channel.

of the graph represent decoding functions. These are iterative algorithms, that proceed as messages are passed between the nodes. Whereas in erasure decoding the messages were discrete $\{0, ?, 1\}$, in sum-product decoding the messages are real numbers from the log domain. In addition, the sum-product algorithm's check node function and variable node function differ from that of erasure decoding.

To express sum-product decoding, the following definitions are used:

- The n variable nodes are indexed $i \in \{1, 2, \ldots, n\}$,
- The m check nodes are indexed $j \in \{1, 2, \ldots, m\}$,
- The input at variable node i is a message L_i calculated from the channel value L_i,
- The message from variable node i to check node j is denoted $Q_{i\to j}$,
- The message from check node j to variable node i is denoted $R_{j\to i}$.

Also, recall that $\mathcal{M}_i$ denotes the set of check nodes that variable node i is connected to, and $\mathcal{N}_j$ denotes the set of variable nodes that check node j is connected to.

The decoder messages L_i, $Q_{i\to j}$ and $R_{j\to i}$ are in the log domain and have the form:

$$\log \frac{\Pr[c_i = 0]}{\Pr[c_i = 1]} \tag{4.5}$$

It is also possible to implement the decoding algorithm in the probability domain of P0 or P1, but log domain messages have several advantages.

Initialization. Initialization has two parts. (1) Given a channel output $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, the input to the sum-product decoder is the log-likelihood ratio:

$$L_i = \log \frac{\Pr[y_i|c_i = 0]}{\Pr[y_i|c_i = 1]} = \log \frac{\Pr[c_i = 0|y_i]}{\Pr[c_i = 1|y_i]}, \tag{4.6}$$

which follows by Bayes rule and $\Pr[c_i = 1] = \Pr[c_i = 0] = \frac{1}{2}$. Given the channel value, y_i the message L_i is easily computed; for the BSC channel see (3.10) and for the AWGN channel see (3.13).

(2) To initialize the decoder messages, the channel message at node i is passed to all the connected check nodes:

$$Q_{i\to j} = L_i \text{ for all } j \in \mathcal{M}_i, \tag{4.7}$$

for $i = 1, 2, \ldots, n$.

Check node function Check node j is connected to d variable nodes. The set of connected variable nodes is $\mathcal{N}_j$ and the degree of this node is $d = |\mathcal{N}_j|$. The check node receives d incoming messages and produces d outgoing messages. Check node j performs the following computations:

$$R_{j\to i} = 2 \tanh^{-1} \left(\prod_{e \in \mathcal{N}_j \setminus i} \tanh \left(\frac{Q_{e\to i}}{2} \right) \right) \text{ for all } i \in \mathcal{N}_j \tag{4.9}$$

Min-Sum Decoding. Min-sum decoding replaces the check node function (4.9) with the following:

$$R_{j\to i} = \left(\prod_{e\in\mathcal{N}_j\setminus i} \text{sign}(Q_{e\to i}) \right) \left(\min_{e\in\mathcal{N}_j\setminus i} |Q_{e\to i}| \right) \tag{4.8}$$

The min operation has lower computational complexity than tanh. Complexity can be further reduced because only the minimum and the second-minimum, over all incoming edges, is needed.

While the min-sum decoder error rates are slightly higher than those for sum-product decoding, the decoding complexity is much lower than that of sum-product decoding, so that min-sum decoding is widely used in practical LDPC decoders, particularly in hardware implementations.

All check nodes $j \in \{1, 2, \ldots, m\}$ perform this operation. (4.9) is the log-domain implementation of APP decoding of the single-parity check code given in Subsection 3.1.4, including application of the sum-product update rule.

Variable node function Variable node i is connected to d check nodes. The set of connected check nodes $\mathcal{M}_i$ and the degree of this node is $d = |\mathcal{M}_i|$. The variable node receives $d+1$ incoming messages: d from the check nodes and one from the channel. It produces d outgoing messages. Variable node i performs the following computations:

$$Q_{i\to j} = L_i + \sum_{e\in\mathcal{M}_i\setminus j} R_{e\to j} \text{ for all } j \in \mathcal{M}_i \tag{4.10}$$

All variable nodes $i \in \{1, 2, \ldots, n\}$ perform this operation. (4.10) is the log-domain implementation of APP decoding of the repeat code given in Subsection 3.1.3, including application of the sum-product update rule.

Hard Decision and Syndrome Check At variable node i, a hard decision is made using all available inputs. It is a variation of the variable node function:

$$t_i = L_i + \sum_{e\in\mathcal{M}_i} R_{e\to i} \tag{4.11}$$

$$\widehat{c}_i = \begin{cases} 0 & t_i \geq 0 \\ 1 & t_i < 0 \end{cases} \tag{4.12}$$

which is similar to 4.10 but no edge is omitted. It is also similar to the MAP decision for the repeat code. All variable nodes $i \in \{1, 2, \ldots, n\}$ perform this operation.

The hard decision vector $\widehat{\mathbf{c}} = [\widehat{c}_1, \widehat{c}_2, \ldots, \widehat{c}_n]$ is a candidate codeword. If $\widehat{\mathbf{c}}\mathbf{H}^{\text{t}} = \mathbf{0}$ then output $\widehat{\mathbf{c}}$ as a valid codeword and decoding stops. If $\widehat{\mathbf{c}}\mathbf{H}^{\text{t}} \neq \mathbf{0}$, then decoding repeats from the check node step. If no convergence is detected after $\ell_{\max}$ iterations, then output the most recent $\widehat{\mathbf{c}}$; it is also possible for the decoder to declare non-convergence after $\ell_{\max}$ unsuccessful iterations.

Sum-product decoding is given in Algorithm 4.1 on page 88. Matlab source code is in Fig. 4.7 on page 87.

4.3.2 Example

Consider the code with parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.13}$$

A codeword $\mathbf{c}$ from this code was transmitted over the binary symmetric channel with $p = 0.1751$ and the sequence:

$$\mathbf{y} = (0, 1, 0, 1, 0, 1) \tag{4.14}$$

was received. Perform sum-product decoding to find the estimated codeword $\widehat{\mathbf{c}}$.

Initialization (1) For the BSC in general:

$$\Pr[\mathsf{Y} = 1|\mathsf{X} = 0] = \Pr[\mathsf{Y} = 0|\mathsf{X} = 1] = p \tag{4.15}$$
$$\Pr[\mathsf{Y} = 0|\mathsf{X} = 0] = \Pr[\mathsf{Y} = 1|\mathsf{X} = 1] = 1 - p, \tag{4.16}$$

so that:

$$L = \begin{cases} \log \frac{1-p}{p} & \text{if } y = 0 \\ \log \frac{p}{1-p} & \text{if } y = 1 \end{cases} \tag{4.17}$$

Specifically for $p = 0.1751$ and $\mathbf{y} = (0, 1, 0, 1, 0, 1)$ the channel messages L_i are:

$$(L_1, L_2, L_3, L_4, L_5, L_6) = (1.55, -1.55, 1.55, -1.55, 1.55, -1.55) \tag{4.18}$$

(2) These messages are passed on the variable-to-check edges:

$$\begin{aligned} Q_{1\to1} &= 1.55 & Q_{1\to2} &= 1.55 \\ Q_{2\to1} &= -1.55 & Q_{2\to3} &= -1.55 \\ Q_{3\to1} &= 1.55 & Q_{3\to3} &= 1.55 \\ Q_{4\to2} &= -1.55 & Q_{4\to4} &= -1.55 \\ Q_{5\to2} &= 1.55 & Q_{5\to4} &= 1.55 \\ Q_{6\to3} &= -1.55 & Q_{6\to4} &= -1.55 \end{aligned}$$

Check node function Consider the operation at check node 1, which has three incoming messages $Q_{1\to1}$, $Q_{2\to1}$, $Q_{3\to1}$. The check node computes three outputs

using (4.9):

$$\begin{aligned}
R_{1\to1} &= 2\tanh^{-1}\left(\tanh\left(\frac{Q_{2\to1}}{2}\right)\cdot\tanh\left(\frac{Q_{3\to1}}{2}\right)\right)\\
&= 2\tanh^{-1}\left(\tanh\left(\frac{-1.55}{2}\right)\cdot\tanh\left(\frac{1.55}{2}\right)\right)\\
&= -0.9\\
R_{1\to2} &= 2\tanh^{-1}\left(\tanh\left(\frac{Q_{1\to1}}{2}\right)\cdot\tanh\left(\frac{Q_{3\to1}}{2}\right)\right)\\
&= 2\tanh^{-1}\left(\tanh\left(\frac{1.55}{2}\right)\cdot\tanh\left(\frac{1.55}{2}\right)\right)\\
&= 0.9\\
R_{1\to3} &= 2\tanh^{-1}\left(\tanh\left(\frac{Q_{1\to1}}{2}\right)\cdot\tanh\left(\frac{Q_{2\to1}}{2}\right)\right)\\
&= 2\tanh^{-1}\left(\tanh\left(\frac{1.55}{2}\right)\cdot\tanh\left(\frac{-1.55}{2}\right)\right)\\
&= -0.9
\end{aligned}$$

In a similar way, all of the check-to-variable messages are found:

$$\begin{array}{lll}
R_{1\to1} = -0.9 & R_{1\to2} = 0.9 & R_{1\to3} = -0.9\\
R_{2\to1} = -0.9 & R_{2\to4} = 0.9 & R_{2\to5} = -0.9\\
R_{3\to2} = -0.9 & R_{3\to3} = 0.9 & R_{3\to6} = -0.9\\
R_{4\to4} = -0.9 & R_{4\to5} = 0.9 & R_{4\to6} = -0.9
\end{array}$$

Variable node function Consider the operation at variable node 1, which has two incoming messages $R_{1\to1}$ and $R_{3\to1}$ and the channel message L_1. The variable node computes two outputs using (4.10):

$$\begin{aligned}
Q_{1\to1} &= L_1 + R_{2\to1}\\
&= 1.55 + (-0.9)\\
&= 0.65\\
Q_{1\to2} &= L_1 + R_{1\to1}\\
&= 1.55 + (-0.9)\\
&= 0.65
\end{aligned}$$

In a similar way, all of the variable-to-check messages are found:

$$\begin{array}{ll}
Q_{1\to1} = 0.6491 & Q_{1\to2} = 0.6491\\
Q_{2\to1} = -2.451 & Q_{2\to3} = -0.6491\\
Q_{3\to1} = 2.451 & Q_{3\to3} = 0.6491\\
Q_{4\to2} = -2.451 & Q_{4\to4} = -0.6491\\
Q_{5\to2} = 2.451 & Q_{5\to4} = 0.6491\\
Q_{6\to3} = -2.451 & Q_{6\to4} = -2.451
\end{array}$$

Hard Decision Consider the hard decision at variable node 1, which which has two incoming messages $R_{1\to1}$ and $R_{3\to1}$ and the channel message L_1. The variable node computes:

$$\begin{aligned} t_1 &= L_1 + R_{1\to1} + R_{2\to1} \\ &= 1.55 + (-0.9) + (-0.9) \\ &= -0.25 \end{aligned}$$

which gives the hard estimate $\widehat{\mathbf{c}}_1 = 1$. In a similar way, all of the hard decisions are found:

$$\begin{aligned} t_1 &= -0.2518 & \widehat{c}_1 &= 1 \\ t_2 &= -1.55 & \widehat{c}_2 &= 1 \\ t_3 &= 1.55 & \widehat{c}_3 &= 0 \\ t_4 &= -1.55 & \widehat{c}_4 &= 1 \\ t_5 &= 1.55 & \widehat{c}_5 &= 0 \\ t_6 &= -3.352 & \widehat{c}_6 &= 1 \end{aligned}$$

Giving the estimated codeword $\widehat{\mathbf{c}} = (1, 1, 0, 1, 0, 1)$.

Check Compute $\widehat{\mathbf{c}} \cdot \mathbf{H}^{\mathrm{t}}$ and find that this is equal to $\mathbf{0}$. Thus, $\widehat{\mathbf{c}}$ is a valid codeword and the sum-product decoding algorithm outputs $\widehat{\mathbf{c}}$ and stops.

4.4 Encoding LDPC Codes

Section 2.3.3 showed how to perform efficient encoding if $\mathbf{H}$ is in triangular form. But for LDPC codes, it is difficult to find a suitable parity-check matrix which is triangular. However, if $\mathbf{H}$ is in a so-called approximate lower triangular (ALT) form, efficient encoding can still be performed.

4.4.1 Approximate Lower Triangular Encoding

The following technique finds the systematic codeword. Given information $\mathbf{u}$, the codeword is written as:

$$\mathbf{c} = [\mathbf{u}, \mathbf{p}_1, \mathbf{p}_2], \tag{4.19}$$

where $\mathbf{p}_1$ has g symbols and $\mathbf{p}_2$ has $m - g$ symbols. The parity symbols $\mathbf{p}_1$ and $\mathbf{p}_2$ are found by solving:

$$\mathbf{H}\mathbf{c}^{\mathrm{t}} = \mathbf{0}. \tag{4.20}$$

Suppose that $\mathbf{H}$ can be written as an approximately lower triangular matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}. \tag{4.21}$$

as shown in Fig. 4.6, where $\mathbf{T}$ is a lower-triangular matrix, and

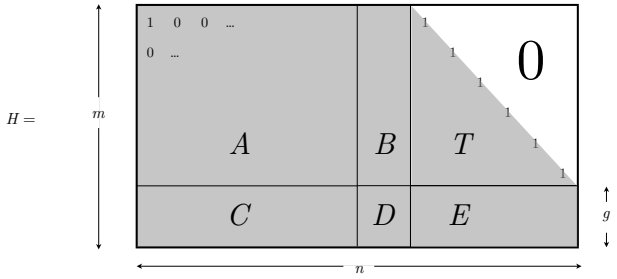


Figure 4.6: Parity-check matrix $\mathbf{H}$ is partially lower-triangular form.

- $\mathbf{T}$ is $(m-g) \times (m-g)$
- $\mathbf{A}$ is $(m-g) \times (n-m)$
- $\mathbf{B}$ is $(m-g) \times g$
- $\mathbf{E}$ is $g \times (m-g)$
- $\mathbf{C}$ is $g \times (n-m)$
- $\mathbf{D}$ is $g \times g$

The term g is called the "gap." If $g = 0$, then encoding by back-substitution is sufficient as described in Subsection 2.3.3. But even if $g > 0$, parity bits $\mathbf{p}_2$ can be encoded by back substitution, if we have already found the parity bits $\mathbf{p}_1$.

Multiply $\mathbf{H}$ on the left by:

$$\begin{bmatrix} \mathbf{I}_{m-g} & \mathbf{0} \\ -\mathbf{E}\mathbf{T}^{-1} & \mathbf{I}_g \end{bmatrix} \tag{4.22}$$

to obtain:

$$\widetilde{\mathbf{H}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \Gamma & \Phi & \mathbf{0} \end{bmatrix} \tag{4.23}$$

where $\Gamma = -\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C}$ and $\Phi = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$. Multiplying on the left is equivalent to performing row operations — $\mathbf{H}$ and $\widetilde{\mathbf{H}}$ are two parity-check matrices for the same code.

The parity-check equations $\widetilde{\mathbf{H}}\mathbf{c}^{\mathrm{t}} = \mathbf{0}$ become:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \Gamma & \Phi & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u}^{\mathrm{t}} \\ \mathbf{p}_1^{\mathrm{t}} \\ \mathbf{p}_2^{\mathrm{t}} \end{bmatrix} = \mathbf{0} \tag{4.24}$$

First, solve the bottom matrix row $\Gamma\mathbf{u}^{\mathrm{t}} + \Phi\mathbf{p}_1^{\mathrm{t}} = \mathbf{0}$ as:

$$\mathbf{p}_1^{\mathrm{t}} = -\Phi^{-1}\Gamma\mathbf{u}^{\mathrm{t}} \tag{4.25}$$

Then, since $\mathbf{u}$ and $\mathbf{p}_1$ are known, $\mathbf{p}_2$ can be found by solving $\mathbf{A}\mathbf{u}^{\mathrm{t}}+\mathbf{B}\mathbf{p}_1^{\mathrm{t}}+\mathbf{T}\mathbf{p}_2^{\mathrm{t}} = \mathbf{0}$. Since $\mathbf{T}$ is lower triangular, $\mathbf{p}_2$ can be found by back-substitution[1]:

$$\mathbf{T}\mathbf{p}_2^{\mathrm{t}} = \mathbf{w}, \tag{4.27}$$

where $\mathbf{w} = -\mathbf{A}\mathbf{u}^{\mathrm{t}} - \mathbf{B}\mathbf{p}_1^{\mathrm{t}}$.

Note that $\mathbf{H}$ should be full rank. Even if $\mathbf{H}$ is full rank, Φ may not be invertible. In this case, swap columns in the left block and center block so that Φ is invertible. Swapping columns changes the position of the code bits.

4.5 Exercises

4.1 Show that if a regular $m \times n$ parity-check matrix $\mathbf{H}$ has d_{v} is even, then the rank cannot be m, that is $\mathrm{rank}(\mathbf{H}) \leq m-1$.

4.2 *Gallager's check node function* For $\beta > 0$, define $f(\beta)$ as:

$$f(\beta) = \log \frac{e^{\beta}+1}{e^{\beta}-1} \tag{4.28}$$

(a) Show that $f^{-1}(\beta) = f(\beta)$.
(b) Show that the $d = 3$ check node function can be written as:

$$r_3 = \mathrm{sign}(q_1)\mathrm{sign}(q_2) f\Big(f(|q_1|) + f(|q_2|)\Big) \tag{4.29}$$

4.3 Consider the code with parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{4.41}$$

Perform erasure decoding with the received sequences:

(a) $\mathbf{y}_1 = [0, 0, 1, ?, ?, ?]$
(b) $\mathbf{y}_2 = [?, ?, ?, 1, 0, 1]$
(c) $\mathbf{y}_3 = [?, ?, ?, 0, 1, ?]$

For each one, give the estimated codeword $\widehat{\mathbf{c}}$, or "failure to decode"

4.4 For the code with parity-check matrix:

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

find systematic codewords corresponding to information sequences:

[1] Alternatively, if the matrices are not too large, $\mathbf{p}_2$ may be found by using the inversion of $\mathbf{T}$:

$$\mathbf{p}_2^{\mathrm{t}} = -(\mathbf{u}\mathbf{A}^{\mathrm{t}} + \mathbf{p}_1\mathbf{B}^{\mathrm{t}})(\mathbf{T}^{\mathrm{t}})^{-1} \tag{4.26}$$

(a) $\mathbf{u}_1 = (1, 0, 0, 1)$.
(b) $\mathbf{u}_2 = (0, 0, 0, 1)$.
(c) $\mathbf{u}_3 = (1, 1, 1, 1)$,

4.5 Using the code with parity-check matrix in eqn. (4.26) , perform sum-product decoding to find the estimated codeword $\widehat{\mathbf{c}}$ when:

(a) On the BSC with $p = 0.21$ the following $\mathbf{y}$ is received:

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

(b) On the AWGN channel with $\sigma^2 = 1$ and transmit mapping $0 \to +1$ and $1 \to -1$, the following $\mathbf{y}$ is received:

$$\mathbf{y} = \begin{bmatrix} -1.2 & -0.8 & 0.6 & 1 & -1.2 & -0.8 \end{bmatrix}.$$

```
function chat = ldpc_decoder(H,Y,ellmax)
[m,n] = size(H);
M     = cell(1,n);
N     = cell(1,m);
for j = 1:m
    N{j} = find( H(j,:) == 1 );
end
for i = 1:n
    M{i} = find( H(:,i) == 1 )'; %transpose is important!
end

%initialize
%R(j,i) is the message r_{j -> i}, Q(j,i) is Q_{i -> j}
R    = zeros(m,n);
Q    = zeros(m,n);
chat = zeros(1,n);
for i = 1:n
    for k = M{i}
        Q(k,i) = Y(i);
    end
end

%begin iterations
for ell = 1:ellmax
    %check node
    for j = 1:m
        for i = N{j}
            e = setdiff(N{j},i);%sum-product update rule
            R(j,i) = 2 * atanh( prod(tanh(Q(j,e) / 2)) );
        end
    end

    %variable node
    for i = 1:n
        for j = M{i}
            e = setdiff(M{i},j);%sum-product update rule
            Q(j,i) = Y(i) + sum( R(e,i) );
        end
    end

    %hard decisions
    for i = 1:n
        e = M{i};
        t(i) = Y(i) + sum( R(e,i) );
        chat(i) = (t(i) < 0);
    end

    %syndrome check
    s = mod(chat * H',2);
    if all(s == 0)
        break;
    end
end
```

Figure 4.7: Matlab source code implementing sum-product decoding.

Algorithm 4.1 Sum-Product Decoding of LDPC Codes

Require: Received sequence $\mathbf{y}$, parity-check matrix $\mathbf{H}$, maximum number of iterations $\ell_{\max}$.

Ensure: Estimated codeword $\widehat{\mathbf{c}}$.

Initialize

for $i = 1, \ldots, n$ **do**

$l_i = \dfrac{2}{\sigma^2} y_i$

$Q_{i \to j} = L_i$ for each $j \in \mathcal{M}_i$.

end for

for $\ell = 1, \ldots, \ell_{\max}$ **do**

Check nodes

for $j = 1, \ldots, m$ **do**

for $i \in \mathcal{N}_j$ **do**

$$R_{j \to i} = 2 \tanh^{-1} \left(\prod_{e \in \mathcal{N}_j \setminus i} \tanh \left(\frac{Q_{e \to j}}{2} \right) \right)$$

end for

end for

Variable nodes

for $i = 1, \ldots, n$ **do**

for $j \in \mathcal{M}_i$ **do**

$$Q_{i \to j} = L_i + \sum_{e \in \mathcal{M}_i \setminus j} R_{e \to i}$$

end for

end for

Hard decisions

for $i = 1, \ldots, n$ **do**

$$t_i = L_i + \sum_{e \in \mathcal{M}_i} R_{e \to i}$$

$$\widehat{c}_i = \begin{cases} 0 & t_i \geq 0 \\ 1 & t_i < 0 \end{cases}$$

end for

Check

if $\widehat{\mathbf{c}} \cdot \mathbf{H}^{\mathrm{t}} = \mathbf{0}$ **then**

Goto End

end if

end for

End: Output estimated codeword $\widehat{\mathbf{c}}$.

Algorithm 4.2 ALT Encoding of LDPC Codes

Require: Full-rank matrix $\mathbf{H}$, with invertible Φ, partitioned according to (4.21). Information sequence $\mathbf{u}$.

Ensure: Systematic codeword satisfying $\mathbf{cH}^{\mathrm{t}}$.

Initialize

Find $\Gamma = -\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C}$

Find $\Phi = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$

Encode

$\mathbf{p}_1^{\mathrm{t}} = -\Phi^{-1}\Gamma\mathbf{u}^{\mathrm{t}}$

$\mathbf{w} = -\mathbf{A}\mathbf{u}^{\mathrm{t}} - \mathbf{B}\mathbf{p}_1^{\mathrm{t}}$

Solve $\mathbf{T}\mathbf{p}_2 = \mathbf{w}$ by back-substitution

Output codeword $\mathbf{c} = [\mathbf{u}, \mathbf{p}_1, \mathbf{p}_2]$.

Chapter 5

Design of LDPC Codes

5.1 Gallager LDPC codes

Given code parameters n, d_{v} and d_{c}, Gallager provided a construction that gives an n-by-m sparse parity-check matrix $\mathbf{H}$ with row degree d_{c}, column degree d_{v} and $m = nd_{\mathsf{v}}/d_{\mathsf{c}}$. The design rate is $R_{\mathrm{d}} = 1 - \frac{d_{\mathsf{v}}}{d_{\mathsf{c}}}$. This is a regular LDPC code. Let $\mathbf{H}_1$ be the matrix consisting of the concatenation of d_{c} identity matrices:

$$\mathbf{H}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \end{bmatrix} \tag{5.1}$$

where $\mathbf{I}$ is the $\frac{n}{d_{\mathsf{v}}} \times \frac{n}{d_{\mathsf{v}}}$ identity matrix. The row weight of $\mathbf{H}_1$ is d_{c} and the column weight is 1.

Let π be a pseudo-random[1] column permutation. Then the parity check matrix for the Gallager LDPC construction is:

$$\mathbf{H} = \begin{bmatrix} \pi_1(H_1) \\ \pi_2(H_1) \\ \vdots \\ \pi_{d_{\mathsf{v}}}(H_1) \end{bmatrix}. \tag{5.2}$$

Each of $\pi_1, \pi_2, \ldots,$ are distinct column permutations. So $\mathbf{H}$ has row weight d_{c} and column weight d_{v}.

Example 5.1. Following Gallager, choose π_1 so that $\pi_1(\mathbf{H}_1)$ forms a staircase pattern, although the choice of π_1 is arbitrary. A Gallager construction parity-

[1] Pseudo-random means that it looks random but is "known." Quite often pseudo-random means a random-looking permutation that is known to both the encoder and decoder.

check matrix $\mathbf{H}$ with $n = 20, d_\mathsf{v} = 3, d_\mathsf{c} = 4$ is:

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

This code has design rate $R_\mathrm{d} = \frac{1}{4}$.

While the design rate is $1 - d_\mathsf{v}/d_\mathsf{c}$, the actual rate may be higher because of linearly dependent rows in the randomly-generated matrix. The rank of an m-by-n parity-check matrix is the number of linearly independent rows. A full-rank matrix has rank m. But if there are linearly dependent rows, then the rank is less than m. Then, the code rate is:

$$R = 1 - \frac{\text{rank H}}{n} \geq 1 - \frac{m}{n} = R_\mathrm{d}, \tag{5.3}$$

that is the code rate $R \geq R_\mathrm{d}$, where R_d is the design rate.

5.2 Irregular LDPC Codes

5.2.1 Degree Distribution

In an irregular LDPC code, the row weight and column weight is not constant. Distributions are used to describe the irregular structure of the parity check matrix. The *node perspective* expresses the fraction of nodes with a given degree. The *edge perspective* expresses the fraction of edges connecting to nodes of a given degree.

For the node perspective, R_j is the fraction of nodes of degree j. Likewise, L_i is the fraction of nodes of degree i. It is convenient to express these distributions as polynomials:

$$R(x) = \sum_{j=1}^{d_\mathsf{c}^\mathrm{max}} R_j x^j \text{ and } L(x) = \sum_{i=1}^{d_\mathsf{c}^\mathrm{max}} L_i x^i. \tag{5.4}$$

For the edge perspective, ρ_j is the fraction of edges that connect to a check node of degree j. Likewise, λ_i is the fraction of edges that connect to a variable node of degree i. The sum of the ρ_j is 1 and the sum of the λ_i is also 1. It is convenient to express these distributions as polynomials:

$$\rho(x) = \sum_{j=1}^{d_c^{\max}} \rho_j x^{j-1} \text{ and } \lambda(x) = \sum_{i=1}^{d_c^{\max}} \lambda_i x^{i-1}. \tag{5.5}$$

Note that in $\rho(x)$, the degree of ρ_j is x^{j-1}, whereas in $R(x)$ the degree of R_j is x^j; similarly for the variable node.

The node perspectives and edge perspectives are related by :

$$\rho(x) = \frac{R'(x)}{R'(1)} \tag{5.6}$$

$$\lambda(x) = \frac{L'(x)}{L'(1)} \tag{5.7}$$

The design rate R_d is:

$$R_\mathrm{d} = 1 - \frac{L'(1)}{R'(1)} = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} \tag{5.8}$$

Example 5.2. Find $R(x)$, $L(x)$ and $\rho(x)$, $\lambda(x)$ for the following parity-check matrix:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

There are 3 variable nodes of degree 1, 3 variable nodes of degree 2 and 1 variable node of degree 3. Since there are 7 nodes in total:

$$L(x) = \frac{3}{7}x + \frac{3}{7}x^2 + \frac{1}{7}x^3 \tag{5.10}$$

All check nodes have degree 4, so $R(x) = x^4$. For the edge perspective, there are 3 edges connected to degree 1 nodes, 6 edges connected to a degree 2 nodes and 3 edges connected to degree 3 nodes. Since there are 12 edges in total:

$$\lambda(x) = \frac{3}{12} + \frac{6}{12}x + \frac{3}{12}x^2 \tag{5.11}$$

All edges are connected to check nodes of degree 4, so $\rho(x) = x^3$.

5.2.2 Code Ensemble

In order to perform analysis of LDPC codes, we consider not a single code, but a set or *ensemble* of codes. All codes in the ensemble have the same degree distribution. They differ in how the variable nodes and check nodes are connected.

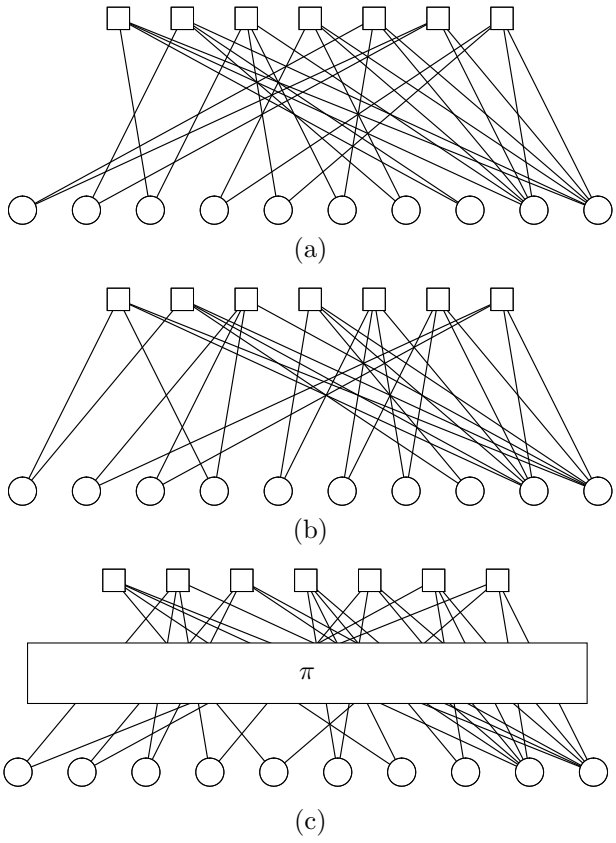


Figure 5.1: (a) One possible LDPC code (b) Another possible LDPC code (c) Ensemble of LDPC codes represented by all possible permutations of edges.

Consider for example, the node degree distribution:

$$L(x) = \frac{8}{10}x^2 + \frac{2}{10}x^6 \text{ and } R(x) = x^3. \tag{5.12}$$

The Tanner graph of one possible $n = 10$ code with this degree distribution is shown in Fig. 5.1-(a). Another possible code with the same degree distribution is shown in Fig. 5.1. Clearly, there are a large possible number of LDPC codes with the same degree distribution.

Now consider to consider not a single code, but the set of all possible codes. Each edge coming from all the variable nodes can be thought of as a "socket". Likewise each edge coming from all the check nodes is also a socket. A pseudorandom permutation connects each variable node socket to each check node socket. For a given degree distribution the *code ensemble* consists of all codes obtained from all distinct permutations, such that there are no cycles of length 2.

When n is small, the structure of the permutation strongly influences the code — some choices of the permutation are good, and some are bad. For example, some permutations may have a large number of short cycles (i.e 4 cycles, 6 cycles), which reduce the performance of the decoder. But as $n \to \infty$ the permutation has less influence on the code. While *some* members of the ensemble will have short cycles, as n gets large, the probability that the members of the ensemble will have a short cycle becomes smaller.

The LDPC decoder for some bit x_i can be seen to be a tree, but only for a few iterations. As the number of iterations increases, the bit x_i will appear again, and the decoder is no longer a tree. But as $n \to \infty$ the number of short cycles decreases, and the probability that x_i appears in the decoder tree decreases. This allows us to ignore loops in the graph, and assume that the decoder has a tree-like structure. Now, we can evaluate the LDPC code using the degree distribution only, and ignore the structure of the permutation.

5.2.3 Density Evolution

Density evolution is a tool to answer the question: given a degree distribution and a channel, will an asymptotically long code with this degree distribution converge or not? Consider asymptotically long LDPC codes that is, $n \to \infty$. Only the degree distribution is considered, and not the structure of the permutation.

Density evolution considers the probability distributions on the messages in the decoder. The messages are now considered to be random variables. The check-to-variable message is a random variable R and the variable-to-check message is a random variable Q. These distributions are:

$$g(r) = \Pr(\mathsf{R} = r) \tag{5.13}$$
$$f(q) = \Pr(\mathsf{Q} = q) \tag{5.14}$$

It is assumed that on any given iteration, all the check-to-variable messages are independent and identically distributed — that is, they have the same distribu-

tion, and are independent. Likewise, on any given iteration, all the variable-to-check messages are independent and identically distributed. Rather than one message for each edge, density evolution uses a single distribution, representing all edges.

Density evolutions uses the degree distribution of the code. Density evolution assumes that the all-zeros codeword was transmitted. This analysis is valid for the BEC, BSC and AWGN channels because these channels satisfy a certain symmetry condition.

Density evolution follows the steps of the iterative decoder, replacing messages with their probability densities. An input to density evolution is the channel parameter ϵ (for BEC), p (for BSC) or σ^2 (for AWGN).

1. Initialization. The channel message is L. Let its distribution be given by $h(l) = \Pr(L = l)$, which is given by:

BEC channel For a BEC with erasure probability ϵ:

$$Q(q) = \begin{cases} 1-\epsilon & q = 0 \\ \epsilon & q =? \\ 0 & q = 1 \end{cases} \tag{5.15}$$

$Q(1) = 0$ because the all zeros codeword was transmitted

BSC channel For a BSC channel with error probability p:

$$Q(q) = \begin{cases} 1-p & q = 0 \\ p & q = 1 \end{cases} \tag{5.16}$$

BI-AWGN channel For the binary-input AWGN channel with noise variance σ^2:

$$Q(q) = \mathcal{N}(-\frac{2}{\sigma^2}, \frac{4}{\sigma^2}), \tag{5.17}$$

where $\mathcal{N}(m, v)$ is a Gaussian with mean m, variance v. If the all-zeros codeword is transmitted, then the received $y \sim N(-1, \sigma^2)$. Recall that the receiver multiplies y by $\frac{2}{\sigma^2}$ to obtain log-domain, which is used by the decoder.

This is used to initialize the variable-to-check message f.

2. Check node. A check node of degree d uses the input $Q(q)$ $d-1$ times to produce the output $R_d(r)$. For an irregular code, the output $R(r)$ is obtained by weighting $R_d(r)$ by its edge weight.

3. Variable node. A variable node of degree $d \leq d_{\mathsf{v}}$ uses the input $R(r)$ $d-1$ times to produce the output $Q_d(q)$. For an irregular code, the output $Q(r)$ is obtained by weighting $Q_d(r)$ by its edge weight.

4. Check for convergence. The probability of error is $\Pr(\mathsf{Q} > 0)$ for log-domain decoding, since the all-zeros codeword was transmitted. If the probability error is sufficiently small (for example, 10^{-10}), then declare convergence

and stop. If the number of iterations exceeds some fixed number (for example 200 iterations), then declare non-convergence and stop. Otherwise, go to Step 2.

5.2.4 Density Evolution for the BEC

For the erasure channel,

$$\Pr(\mathsf{Y} = y|\mathsf{X} = 0) = \begin{cases} 1-\epsilon & \text{if } y = 0 \\ \epsilon & \text{if } y =? \end{cases} \tag{5.18}$$

Let y be the probability the check-to-variable message R is erased, and let x be the probability the variable-to-check message Q is erased:

$$y = \Pr(\mathsf{R} = ?) \tag{5.19}$$
$$x = \Pr(\mathsf{Q} = ?) \tag{5.20}$$

Consider a check node of degree d, which has inputs $\mathsf{Q}_1, \ldots, \mathsf{Q}_{d-1}$, all have the same distribution y. The probability that the output R_d is erased can be found using the check node decoding rule from Chapter 4. Then x can be found as:

$$\begin{aligned} y &= \Pr(\text{any of } \mathsf{Q}_1, \mathsf{Q}_2, \ldots, \mathsf{Q}_{d-1} \text{ are } ?) && (5.21)\\ &= 1 - \Pr(\text{none of } \mathsf{Q}_1, \mathsf{Q}_2, \ldots, \mathsf{Q}_{d-1} \text{ are } ?) && (5.22)\\ &= 1 - \Pr(\mathsf{Q}_1 \neq ?)\Pr(\mathsf{Q}_2 \neq ?)\cdots\Pr(\mathsf{Q}_{d-1} \neq ?) && (5.23)\\ &= 1 - (1-x)^{d-1} && (5.24) \end{aligned}$$

For a regular code, $\rho(x) = x^{d-1}$ and (5.24) can be written as

$$y(x) = 1 - \rho(1-x). \tag{5.25}$$

This form is valid for irregular codes as well. For an irregular code, check nodes of different degree d will produce different values for x. These different values are combined by weighting them in proportion to the fraction of edges connected to check nodes of degree d. These weights are given by the polynomial ρ.

Consider a variable node of degree d, which has inputs Y and $\mathsf{R}_1, \ldots, \mathsf{R}_{d-1}$, all have the same distribution y. The probability that the output Q_d is erased can be found using the variable node decoding rule from Chapter 4. Then y can be found as:

$$\begin{aligned} x &= \Pr(q =?) && (5.26)\\ x &= \Pr(\text{all of } \mathsf{Y}, \mathsf{R}_1, \ldots, \mathsf{R}_{d-1} \text{ are } ?) && (5.27)\\ &= \Pr(\mathsf{Y} = ?)\Pr(\mathsf{R}_1 = ?)\cdots\Pr(\mathsf{R}_{d-1} = ?) && (5.28)\\ &= \epsilon y^{d-1} && (5.29) \end{aligned}$$

Similarly to the check node, for an irregular code the probability of a degree d edge is weighted according to the proportion of edges, using λ:

$$x(y) = \epsilon\lambda(y). \tag{5.30}$$

Density evolution for the BEC proceeds by tracking the probability that a message is erased as the iterations progress. On iteration ℓ, the check node has input $x^{(\ell)}$ and output $y^{(\ell)}$. The variable node has input $y^{(\ell)}$ and output $x^{(\ell+1)}$. Density evolution is initialized with $x^{(1)} = \epsilon$, the probability of erasure from the channel. Then, the sequence:

$$\epsilon \to y^{(1)} \to x^{(2)} \to y^{(2)} \to x^{(3)} \to y^{(3)} \to \cdots$$

are found by iteratively applying (5.25) and (5.30).

The output of the density evolution procedure is either "converged to 0" or "did not achieve 0 after $\ell_{\max}$ iterations" If during the iterations, $x^{(\ell)} \to 0$, then the probability of erasure went to zero, and the decoder converged to the correct solution 0. On the other hand, if after a large number of iterations $\ell_{\max}$, $x^{(\ell_{\max})} > 0$, then 0 probability of error was not achieved.

Density evolution is illustrated in Fig. 5.2 for a $d_{\mathsf{v}} = 3, d_{\mathsf{c}} = 6$ regular LDPC code so $\lambda(x) = x^2$ and $\rho(x) = x^5$. For the check node, the polynomial $y(x) = 1 - \rho(1 - x)$, is plotted. For $\epsilon = 0.3$, the variable node function $x(y) = \epsilon\lambda(y)$ is shown in Fig. 5.2-(a), with the x-axis and y-axis reversed. The trajectory of decoding is also shown, beginning with $\epsilon = 0.3$, which gives $y^{(1)} = 0.8319$. The trajectory continues iteratively. In this case of $\epsilon = 0.3$, the trajectory reaches 0 after about 6 iterations. Thus, we expect that a long $(3, 6)$ regular LDPC code should have a low probability of decoder error when $\epsilon = 0.3$.

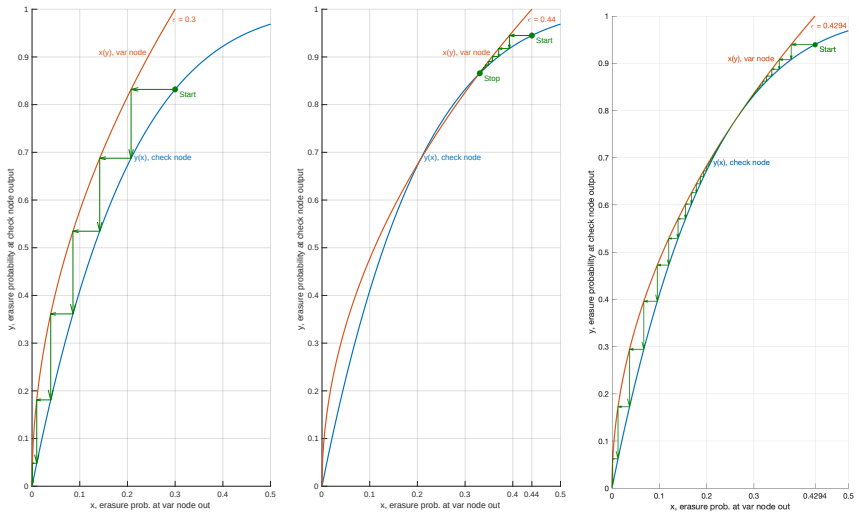
The case of $\epsilon = 0.44$ is shown in Fig. 5.2-(b). In this case, the channel is worse, and lines for the check node and variable node intersect. Because they intersect, the trajectory progresses, but then stops at the point of intersection. The iterations converge to a point which is not 0, and even after a large number of iterations, convergence to 0 is not achieved. Thus, we expect that a long $(3, 6)$ regular LDPC code should have a high probability of decoder error when $\epsilon = 0.3$.

5.2.5 Noise Threshold

We already know that increasing the block length n of a code improves its performance. This section deals with the question: What happens to LDPC codes if we increase block length of an LDPC code to infinity? A phenomenon called the *noise threshold* appears. If the channel noise value is better than the noise threshold parameter, then reliable decoding is possible and the probability of error goes to 0.

For a given channel, density evolution tells us whether or not the decoder will converge to the correct codeword. If the channel is very good, then the decoder should converge. If the channel is very bad (or degraded), then the decoder will not converge. The *noise threshold* is the most degraded channel for which the density evolution converges. It is the worst channel for which reliable decoding is possible as the block length $n \to \infty$.

For the BEC with erasure probability ϵ the noise threshold is the *largest* value ϵ^* such that convergence is obtained . For the code of Fig. 5.2, the noise threshold is $\epsilon^* = 0.4294$. The trajectory for this channel is shown in Fig. 5.2-(c).



(a) (b) (c)

Figure 5.2: Density evolution on BEC for design rate 1/2 LDPC with variable node degree 3, check node degree 6. (a) For BEC erasure probability $\epsilon = 0.3$, the iterations converge to probability of erasure 0 (b) For $\epsilon = 0.44$, the iterations do not converge to 0, thus the probability of error is not zero. (c) $\epsilon = 0.4295$ is the largest value such that convergence to 0 is obtained, and thus is the noise threshold for this code.

It can be seen that the variable node curve nearly touches the check node curve, leaving just enough space for the trajectory to pass through. Although it may not be clear from Fig. 5.2-(c), nearly 500 iterations are needed to converge to 0. The following table shows the noise thresholds for various regular LDPC codes with design rate 1/2.

$(d_\mathsf{v}, d_\mathsf{c})$	R_d	ϵ^*
(2,4)	0.5	0.3333
(3,6)	0.5	0.4294
(4,8)	0.5	0.3834
(5,10)	0.5	0.3415

For the BSC with error probability p, the noise threshold is the *largest* value p^*. For the AWGN with noise variance σ^2, the noise threshold is the *smallest* value $(\sigma^*)^2$.

5.2.6 Irregular LDPC Codes

For the BEC with parameter ϵ, the Shannon capacity is $1 - \epsilon$. For $\epsilon = 0.5$, as the block length $n \to \infty$, there exists a code with $R = 1/2$ for which reliable communication on this channel is possible. From density evolution results in the previous section, a column weight 3, rate 1/2 LDPC code has a noise threshold of 0.4294. This is the most degraded channel for which such an LDPC code

has reliable decoding. From this, we conclude that regular LDPC codes cannot achieve channel capacity.

However, irregular LDPC codes can more closely approach the channel capacity. The table below shows 3 irregular LDPC codes, all with design rate close to 0.5. The best irregular code in the table below has a noise threshold of $\epsilon^* = 0.480977$, which better than the best regular code. This irregular code, while close to channel capacity, does not achieve the channel capacity, a gap of 0.019023 remains.

$\lambda(x)$	$\rho(x)$	R_{d}	ϵ^*
$\frac{1}{10}x + \frac{1}{2}x^2 + \frac{1}{100}x^{10} + \frac{39}{100}x^{19}$	$\frac{1}{2}x^7 + \frac{1}{2}x^8$	0.502034	0.470846
$\lambda_2(x)$	x^5	0.500367	0.480977
$\lambda_3(x)$	x^5	0.498958	0.480721

$$\lambda_2(x) = 0.409x + 0.202x^2 + 0.0768x^3 + 0.1971x^6 + 0.1151x^7$$

$$\lambda_3(x) = 0.416x + 0.166x^2 + 0.1x^3 + 0.07x^4 + 0.053x^5 + 0.042x^6 + 0.035x^7 + 0.03x^8 + 0.026x^9 + 0.023x^{10} + 0.02x^{11} + 0.0183x^{12}$$

5.3 Protograph and Quasi-Cyclic LDPC Codes

Protographs are an effective way to construct LDPC codes. Many of the codes used in communication standards use LDPC codes based on protographs (5G New Radio, WiFi IEEE 802.11n, WiMax IEEE 802.16e, etc.). Quasi-cyclic LDPC codes are a specific type of protograph-based codes.

5.3.1 Protograph Construction

Protograph codes are constructed by repeating a small base graph several times, and then permuting the edges between the copies in a specific way. In particular begin with a small bipartite graph. Duplicate this graph so there are z independent copies. Consider any edge e in the original graph. In the repeated graph e appears z times, and they are connected to z check nodes. To form the protograph code, the connection of the z edges to z check nodes is permuted. This is repeated for all edges in the graph. This process of constructing a graph from a protograph is called lifting.

Example 5.3. Lift the following base graph with $z = 3$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{5.31}$$

Begin by making 3 copies of the base graph, shown in Fig. 5.3-(a). First, the edges connecting variable node 1 and check node 1 are permuted, as shown in

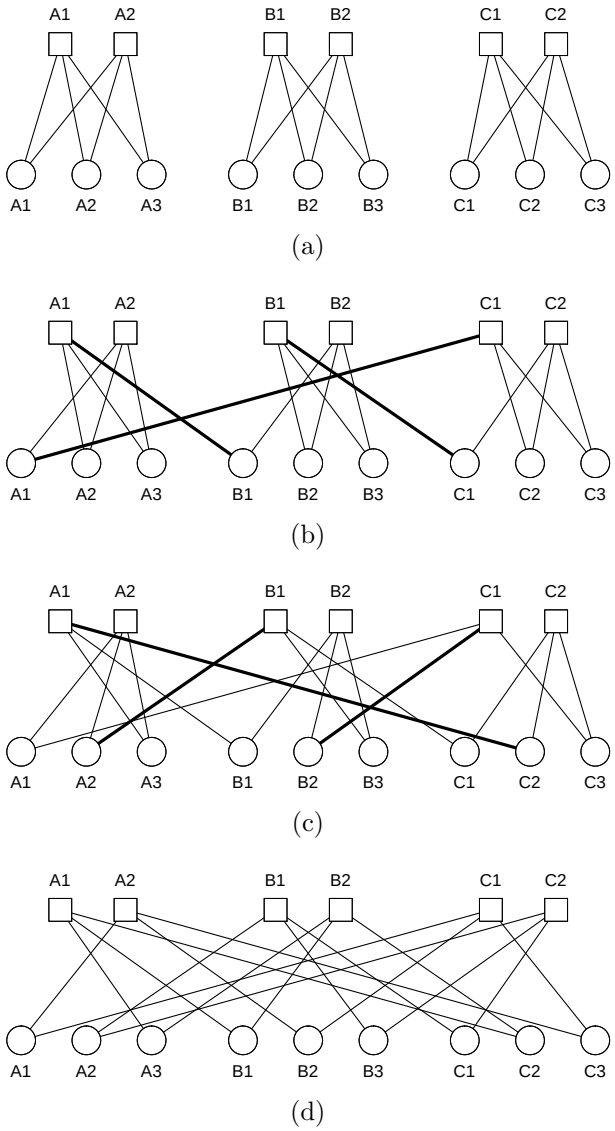


Figure 5.3: Protograph construction for Example 5.3. (a) Base graph repeated $z = 3$ times. (b) The edges connecting variable node 1 and check node 1 are permuted. (c) The edges connecting variable node 2 and check node 1 are permuted. (d) Fully permuted protograph.

Fig. 5.3-(b). This continues for the edges connecting variable node 2 and check node 1, as shown in Fig. 5.3-(c). A final graph with all edges permuted is shown in Fig. 5.3-(d).

The base graph and the lifted graph have a parity-check matrix. Let $\mathbf{H}_{\text{base}}$ be the protograph M-by-N matrix containing 0s and 1s, then parity-check matrix $\mathbf{H}$ for the lifted code has size zM-by-zN. The process of constructing $\mathbf{H}$ from $\mathbf{H}_{\text{base}}$ is called lifting.

After repeating the base graph $z = 3$ times, the lifted graph shown in Fig. 5.3-(a) corresponds to matrix:

$$\begin{array}{c} \begin{array}{cccccccccc} \text{A1} & \text{B1} & \text{C1} & \text{A2} & \text{B2} & \text{C2} & \text{A3} & \text{B3} & \text{C3} & \end{array} \\ \left[\begin{array}{ccc|ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \begin{array}{c} \text{A1} \\ \text{B1} \\ \text{C1} \\ \\ \text{A2} \\ \text{B2} \\ \text{C2} \end{array} \end{array}$$

The rows and columns of the matrix have been labeled to show the correspondence with the variable nodes and check nodes, because the order is not the same.

After applying all permutations, the lifted graph shown in Fig. 5.3-(d) has matrix:

$$\begin{array}{c} \begin{array}{cccccccccc} \text{A1} & \text{B1} & \text{C1} & \text{A2} & \text{B2} & \text{C2} & \text{A3} & \text{B3} & \text{C3} & \end{array} \\ \left[\begin{array}{ccc|ccc|ccc} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \begin{array}{c} \text{A1} \\ \text{B1} \\ \text{C1} \\ \\ \text{A2} \\ \text{B2} \\ \text{C2} \end{array} \end{array}$$

The permutation applied for each edge can be seen in each 3-by-3 block shown of the matrix.

5.3.2 Cyclic Permutation Matrix

A $z \times z$ *right-shift cyclic permutation matrix* $\mathbf{P}$ has the property that:

$$(x_q, x_1, x_2, \ldots, x_{q-1}) = (x_1, x_2, \ldots, x_q) \cdot \mathbf{P} \tag{5.32}$$

For example, if $z = 5$ then:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.33}$$

Note that $\mathbf{P}^2$ is shift-by-two:

$$\mathbf{P}^2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$\mathbf{P}^3$ is shift-by-three, etc. Also $\mathbf{P}^0 = \mathbf{I}_q$, the $q \times q$ identity matrix, that is, no shift. If $t = i \bmod q$, then $\mathbf{P}^i$ is shift-by-t.

5.3.3 Quasi-Cyclic LDPC Codes

A quasi-cyclic LDPC code is a protograph code where the edge permutations are cyclic permutations. If the permutations correspond to a cyclic permutation, then we have a *quasi-cyclic LDPC code.* A protograph code where the random permutation is a permutation matrix is a *quasi-cyclic code.*

The 1s in the protograph matrix are replaced by z-by-z permutation matrices to obtain a zM by zN parity-check matrix $\mathbf{H}$. In the matrix representation of the lifted code, each block was represented by a $z \times z$ permutation. For QC-LDPC codes, we work directly with a parity-check matrix $\mathbf{H}$ formed from permutation matrices. The parity-check matrix for a QC-LDPC code is given by:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^{p_{1,1}} & \mathbf{P}^{p_{1,2}} & \cdots & \mathbf{P}^{p_{1,N}} \\ \mathbf{P}^{p_{2,1}} & \mathbf{P}^{p_{2,2}} & \cdots & \mathbf{P}^{p_{2,N}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{p_{M,1}} & \mathbf{P}^{p_{M,2}} & \cdots & \mathbf{P}^{p_{M,N}} \end{bmatrix}, \tag{5.34}$$

where $p_{j,i} \leq 0$ or $p_{j,i} = -1$. If $p \geq 0$, then $\mathbf{P}^p$ is the usual shift-by-p right-cyclic shift operation. However, it may be necessary to designate an all-zeros matrix, and thus $p = -1$ is thus reserved, that is $\mathbf{P}^{-1} \stackrel{\text{def}}{=} \mathbf{0}$, the $z \times z$ all-zeros matrix.

In the example above, a 2×3 base graph was lifted with $z = 3$ to form a 6×9 code using cyclic permutations. The code corresponding to the final graph is given by:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^1 & \mathbf{P}^2 & \mathbf{P}^0 \\ \mathbf{P}^0 & \mathbf{P}^1 & \mathbf{P}^2 \end{bmatrix} \tag{5.35}$$

where

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}. \tag{5.36}$$

To represent a QC-LDPC code, we need only the powers of $\mathbf{P}$ and z. So the above code can be represented by:

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} \tag{5.37}$$

and $z = 3$. This compact representation is useful for large LDPC codes.

For larger base matrices, it is necessary to specify some block matrices as all-zeros blocks. Consider the 4×12 base matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

When lifted, positions with a 1 need a permutation, indicated by a power of $\mathbf{P}$. To indicate the zeros matrix, -1 is written. For example, here is the lifted matrix:

$$\begin{bmatrix} 0 & 6 & 13 & -1 & 11 & 1 & -1 & 22 & 21 & 2 & 21 & 13 \\ -1 & 1 & -1 & 9 & 10 & 11 & 3 & 19 & 11 & -1 & 10 & 5 \\ 12 & 20 & 20 & 1 & 16 & -1 & 5 & -1 & 7 & 6 & 22 & 15 \\ 12 & -1 & 12 & 3 & -1 & 9 & 3 & 2 & 2 & 5 & 13 & 22 \end{bmatrix}$$

Here, 0 means $\mathbf{P}^0 = \mathbf{I}_z$, while -1 means the $z \times z$ all-zeros matrix $\mathbf{0}$. Note that the degree distribution of the base matrix is the same as the lifted matrix. This is important for designing irregular QC-LDPC codes.

5.3.4 Cycles in QC-LDPC Codes

Consider cycles in the graph. If the base matrix has no cycles of length l, then the lifted graph will also have no cycles of length l. However, even if the base matrix has a cycle of length l, the lifted graph will not necessarily have a cycle of length l.

Consider a protograph which has cycles of length $l = 4$ and length $l = 6$. For $l = 4$, the cycle is:

$$(j_1, i_1) \to (j_1, i_2) \to (j_2, i_2) \to (j_2, i_1) \to (j_1, i_1).$$

where j_1, j_2 are rows and i_1, i_2 are columns. Then, a cycle exists if and only if:

$$p_{j_1,i_1} - p_{j_1,i_2} + p_{j_2,i_2} - p_{j_2,i_1} = 0 \pmod{z} \tag{5.38}$$

For $l = 6$, the cycle is:

$$(j_1, i_1) \to (j_1, i_2) \to (j_2, i_2) \to (j_2, i_3) \to (j_3, i_3) \to (j_3, i_1) \to (j_1, i_1).$$

where j_1, j_2, j_3 are rows and i_1, i_2, i_3 are columns. Then, a cycle exists if and only if:

$$p_{j_1,i_1} - p_{j_1,i_2} + p_{j_2,i_2} - p_{j_2,i_3} + p_{j_3,i_3} - p_{j_3,i_1} = 0 \pmod{z} \tag{5.39}$$

For cycles of length $l = 8$, it is more complicated to identify the cycles. Whereas for $l = 4$ and $l = 6$, each has one protograph pattern which can generate a cycle, for $l = 8$, there are several protograph patterns which can can induce a cycle of length 8.

5.3.5 Design of QC-LDPC Codes

Design of quasi-cyclic LDPC codes:

- Begin with an irregular degree distribution.
- Create a base graph with that degree distribution
- Lift the graph by z to create a long code
- Choose circulant powers to minimize short cycles in the graph.

Chapter 6

Polar Codes

Polar codes were introduced by Erdal Arıkan in 2009. While it has been known for some time that random linear codes can achieve channel capacity, polar codes are the first deterministic construction that can achieve the capacity of binary memoryless symmetric (BMS) channels. They have relatively low decoding complexity, and a flexible construction, in the sense of being able to choose any rate. Polar codes are used for the control channel (as opposed to the data channel) in 5G mobile broadband communications.

6.1 Preliminaries

6.1.1 Kronecker Product

For an $n \times m$ matrix $\mathbf{A}$ and a $p \times q$ matrix $\mathbf{B}$, their *Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ is the $np \times mq$ matrix $\mathbf{A} \otimes \mathbf{B}$ given by:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2m}\mathbf{B} \\ \vdots & & \ddots & \vdots \\ a_{n1}\mathbf{B} & a_{n2}\mathbf{B} & \cdots & a_{nm}\mathbf{B} \end{bmatrix}$$

Powers $s \geq 1$ of a Kroneker matrix are:

$$\mathbf{A}^{\otimes s} = \mathbf{A} \otimes \mathbf{A}^{\otimes s-1} = \underbrace{\mathbf{A} \otimes \mathbf{A} \otimes \cdots \otimes \mathbf{A}}_{s \text{ times}} \tag{6.1}$$

The Kronecker product is associative: $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$. The *mixed product property* states: If $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are matrices of size such that the products $\mathbf{AC}$ and $\mathbf{BD}$ exist then:

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}). \tag{6.2}$$

Wikipedia: Kronecker product

The following matrix $\mathbf{F}^{\otimes n}$ is important in the construction of polar codes:

$$\mathbf{F}^{\otimes n} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n} \tag{6.3}$$

for $n \geq 1$, and $N = 2^n$, the matrix is an $N \times N$ square matrix. This matrix is its own inverse over the binary field: $(\mathbf{F}^{\otimes n})^{-1} = \mathbf{F}^{\otimes n}$.

Example 6.1. The $\mathbf{F}^{\otimes n}$ matrices for $N = 2, 4$ and 8 are:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{6.4}$$

$$\mathbf{F}^{\otimes 2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.5}$$

$$\mathbf{F}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.6}$$

The matrix $\mathbf{F}^{\otimes n}$ is related to the Hadamard matrix. Wikipedia: Hadamard matrix

6.1.2 Reverse Shuffle and Bit-Reversal Permutations

A permutation of a vector is a rearrangement of the elements of that vector. A permutation matrix has one 1 in each column and one 1 in each row; all other entries are 0. A permutation matrix $\mathbf{P}$ is an orthogonal matrix, so $\mathbf{P}^{-1} = \mathbf{P}^{\mathrm{t}}$, that is, the inverse permutation may be found using $\mathbf{P}^{\mathrm{t}}$. Wikipedia: Permutation matrix

---

Example 6.2. The permutation:

$$(1, 2, 3, 4) \to (1, 4, 3, 2) \tag{6.7}$$

Can be induced by multiplying on the right with matrix $\mathbf{P}$:

$$(1, 2, 3, 4) \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{\mathbf{P}} = (1, 4, 3, 2) \tag{6.8}$$

A *perfect shuffle* is a permutation of N elements, $N \geq 4$ and N even, expressed using permutation matrix $\mathbf{S}_N$:

$$(1, 2, 3, 4, \ldots, N) \cdot \mathbf{S}_N = (1, \frac{N}{2} + 1, 2, \frac{N}{2} + 2, 3, \frac{N}{2} + 3, \ldots, \frac{N}{2}, N). \tag{6.9}$$

For example with $N = 8$ the perfect shuffle is:

$$(1, 2, 3, 4, 5, 6, 7, 8) \rightarrow (1, 5, 2, 6, 3, 7, 4, 8). \tag{6.10}$$

For $N = 2$, $\mathbf{S}_2 = \mathbf{I}_2$. Wikipedia: Perfect Shuffle

The *reverse shuffle* $\mathbf{R}_N$ is the inverse of the perfect shuffle:

$$(1, 2, 3, 4, \ldots, N) \cdot \mathbf{R}_N = (1, 3, 5, \ldots, N - 1, 2, 4, 6, \ldots, N), \tag{6.11}$$

For example, with $N = 8$, the reverse shuffle is:

$$(1, 2, 3, 4, 5, 6, 7, 8) \rightarrow (1, 3, 5, 7, 2, 4, 6, 8). \tag{6.12}$$

Since $\mathbf{R}_N$ and $\mathbf{S}_N$ are permutation matrices, $\mathbf{R}_N = \mathbf{S}_N^{-1} = \mathbf{S}_N^{\mathrm{t}}$ and $\mathbf{R}_N \mathbf{S}_N = \mathbf{I}_N$.

Example 6.3. The $N = 8$ reverse shuffle matrix is:

$$\mathbf{R}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.13}$$

which gives the reverse shuffle:

$$(1, 2, 3, 4, 5, 6, 7, 8) \cdot \mathbf{R}_8 = (1, 3, 5, 7, 2, 4, 6, 8) \tag{6.14}$$

The Kronecker product does not commute in general, but if $\mathbf{A}$ has two rows, and $\mathbf{S}_N$ is the perfect shuffle matrix (described in Subsection 6.1.2) then:

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{S}_N^{\mathrm{t}} (\mathbf{B} \otimes \mathbf{A}) \mathbf{S}_N \tag{6.15}$$

A *bit-reversal shuffle* of N numbers is the permutation induced by reversing the order of bits in an n-bit binary representation. The binary string $(d_0 d_1 \ldots d_{n-1})$ represents the integert $\sum_{i=0}^{n-1} 2^i d_i$, where $n = \log_2 N$; d_0 is the LSB (least significant bit) and d_{n-1} is the MSB.

Consider for example $N = 8$. The bits and their reversals are:

0	000	—	000	0
1	100	—	001	4
2	010	—	010	2
3	110	—	011	6
4	001	—	100	1
5	101	—	101	5
6	011	—	110	3
7	111	—	111	7

This induces the permutation $(0,1,2,3,4,5,6,7) \to (0,4,2,6,1,5,3,7)$, which is equivalent to the 1-index permutation:

$$(1,2,3,4,5,6,7,8) \to (1,5,3,7,2,6,4,8) \tag{6.16}$$

The permutation matrix for bit-reversal can be found recursively for $N = 4, 8, 16, 32, \ldots$,

$$\mathbf{B}_N = \mathbf{R}_N \cdot (\mathbf{I}_2 \otimes \mathbf{B}_{N/2}), \tag{6.17}$$

where $\mathbf{B}_2 = \mathbf{I}_2$.

Example 6.4. The $N = 4$ bit-reversal shuffle is:

$$\begin{aligned}\mathbf{B}_4 &= \mathbf{R}_4 \cdot (\mathbf{I}_2 \otimes \mathbf{B}_2) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

since $\mathbf{I}_2 \otimes \mathbf{B}_2 = \mathbf{I}_4$. This is used to find the $N = 8$ bit-reversal shuffle $\mathbf{B}_8$:

$$\begin{aligned}\mathbf{B}_8 &= \mathbf{R}_8 \cdot (\mathbf{I}_2 \otimes \mathbf{B}_4) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 1 & & & & \\ & & & & 1 & 0 & 0 & 0 \\ & & & & 0 & 0 & 1 & 0 \\ & & 0 & & 0 & 1 & 0 & 0 \\ & & & & 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{6.18}$$

This matrix induces the permutation:

$$(1,2,3,4,5,6,7,8) \cdot \mathbf{B}_8 = (1,5,3,7,2,6,4,8), \tag{6.19}$$

which is the permutation in (6.16).

6.2 Polar Codes

6.2.1 Definition

Definition 6.1. For $N = 2^n, n \geq 1$ and $\mathcal{A} \subseteq \{1,2,3,\ldots,N\}$, a *polar code* is the linear block code with codewords generated by the rows indexed by $\mathcal{A}$ in the matrix:

$$\mathbf{G}_N = \mathbf{B}_N \cdot \mathbf{F}^{\otimes n}, \tag{6.20}$$

where $\mathbf{B}_N$ is the bit-reversal permutation matrix. The set $\mathcal{A}$ is the *information set*.

The dimension of a polar code is $K = |\mathcal{A}|$, the cardinality of the set $\mathcal{A}$, and the rate is $R = K/N$. Since $\mathbf{B}_N$ is a permutation matrix, polar codes could have been defined using $\mathbf{F}^{\otimes n}$ alone. However, the ordering induced by the permutation $\mathbf{B}_N$ will simplify the decoding.

Distinct from other linear block codes, for polar codes we write the information vector $\mathbf{u}$ with N elements rather than K elements:

$$\mathbf{u} = (u_1, u_2, u_3, \ldots, u_N). \tag{6.21}$$

Information is placed into the K positions indicated by $\mathcal{A}$. The remaining $N-K$ bits are called *frozen bits*, and are set to be constant. For convenience frozen bits are set to 0, but in general may be set to 0 or 1 arbitrarily, assuming that the encoder and decoder agree on the value of the frozen bits. Because there are K information positions, the dimension of the code remains K. Thus, codeword $\mathbf{c}$ corresponding to $\mathbf{u}$ is

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}_N. \tag{6.22}$$

Example 6.5. Consider the $N = 8$ polar code with information set $\mathcal{A} = \{4,6,7,8\}$. There are $K = |\mathcal{A}| = 4$ information bits so the rate is $R = 0.5$. The information vector is:

$$\mathbf{u} = (0,0,0,u_4,0,u_6,u_7,u_8) \tag{6.23}$$

The generator matrix is $\mathbf{G}_8 = \mathbf{B}_8 \cdot \mathbf{F}^{\otimes 3}$:

$$\mathbf{G}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.24}$$

where $\mathbf{B}_8$ is given by (6.18) and $\mathbf{F}^{\otimes 3}$ is given by (6.6). Codewords can be found from $\mathbf{u} \cdot \mathbf{G}_8$.

Note that due to the frozen bits, 4 rows of $\mathbf{G}_8$ do not contribute to the code, so a codeword $\mathbf{c}$ can also be generated by:

$$\mathbf{c} = [u_4, u_6, u_7, u_8] \cdot \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{6.25}$$

which is the more familiar K-by-N form of the generator matrix. This matrix generates the same code as the first-order Reed-Muller code of Example 2.20, after a permutation of the code bits.

6.2.2 Representation of Polar Codes

A graphical representation of polar codes is useful for encoding, decoding, and understanding polar code's recursive structure. First, define a permutation matrix $\mathbf{P}$ which consists of repeats of the reverse shuffle $\mathbf{R}$. In particular, let $n = \log_2 N$ and define $\mathbf{P}_n^{(\ell)}$ as:

$$\mathbf{P}_N^{(\ell)} = \mathbf{I}_{2^{\ell-1}} \otimes \mathbf{R}_{2^{n-\ell+1}}, \tag{6.26}$$

for $1 \leq \ell \leq n$. When $\ell = 1$ the length of the reverse shuffle is equal to N, so no repetition is necessary and $\mathbf{P}_N^{(1)} = \mathbf{R}_N$. When $\ell = 2$, the reverse shuffle $\mathbf{R}_{N/2}$ is repeated twice. When $\ell = n$, $\mathbf{P}_N^{(\ell)}$ is the identity matrix, that is $\mathbf{P}_N^{(n)} = \mathbf{I}_N$.

The following decomposition of $\mathbf{G}_N$ is a useful representation for encoding and decoding.

Proposition 6.1. The polar code generator matrix $\mathbf{G}_N$ as given in Definition 6.1 can be written as:

$$\mathbf{G}_N = (\mathbf{I}_{N/2} \otimes \mathbf{F}) \prod_{i=1}^{n-1} \left(\mathbf{P}_N^{(i)} \cdot (\mathbf{I}_{N/2} \otimes \mathbf{F}) \right) \tag{6.27}$$

Consider this decomposition for various values of N. When $N = 2$, this is simply:

$$\mathbf{G}_2 = \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{6.28}$$

A block diagram representation is useful for expressing the encoder, the decoder as well as the recursive structure of polar codes. Beginning with $N = 2$, the encoding function $\mathbf{c} = \mathbf{u}\mathbf{G}_2$ with $\mathbf{G}_2 = \mathbf{F}$ is $c_1 = u_1 + u_2$ and $c_2 = u_2$. A block diagram for this encoding is shown in Fig. 6.1. This block is a fundamental unit of polar codes, which will be repeated and used recursively for all larger values of N.

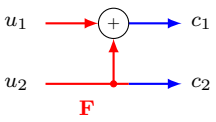


Figure 6.1: Encoder structure when $N = 2$.

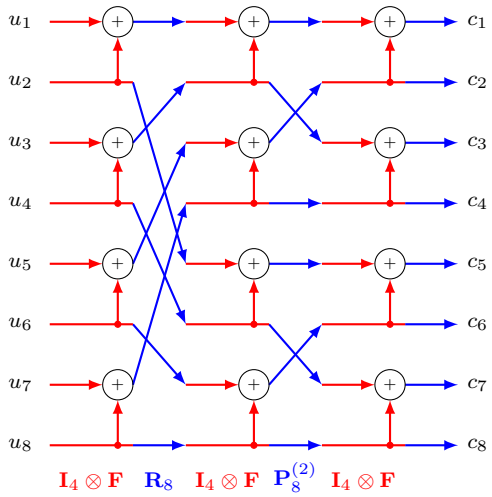


Figure 6.3: Encoder structure for $N = 8$ polar code.

When $N = 4$, the decomposition is $\mathbf{G}_4 = (\mathbf{I}_2 \otimes \mathbf{F}) \cdot \mathbf{R}_4 \cdot (\mathbf{I}_2 \otimes \mathbf{F})$. Using this, the encoding $\mathbf{c} = \mathbf{u}\mathbf{G}_4$ can be obtained by performing three operations in sequence: multiply by $(\mathbf{I}_2 \otimes \mathbf{F})$, then multiply by the reverse shuffle $\mathbf{R}_4$, and then multiply by $(\mathbf{I}_2 \otimes \mathbf{F})$ again. Here, we have $\mathbf{P}_4^{(1)} = \mathbf{R}_4$. The block diagram representation of these operations are shown in Fig. 6.2. Here it can be seen that the fundamental unit $\mathbf{F}$ appears four times in the encoder structure for $N = 4$.

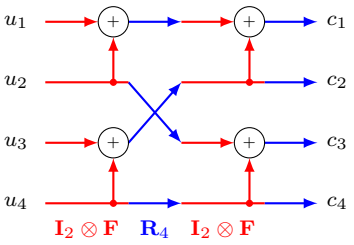


Figure 6.2: Encoder structure when $N = 4$.

When $N = 8$, the decomposition is:

$$\mathbf{G}_4 = (\mathbf{I}_4 \otimes \mathbf{F}) \cdot \mathbf{R}_8 \cdot (\mathbf{I}_4 \otimes \mathbf{F}) \cdot \mathbf{P}_8^{(2)} \cdot (\mathbf{I}_4 \otimes \mathbf{F}) \tag{6.29}$$

Using this, the encoding $\mathbf{c} = \mathbf{u}\mathbf{G}_8$ can be obtained by performing the above operations in sequence. The block diagram representation of these operations are shown in Fig. 6.3. Here, $\mathbf{P}_8^{(1)} = \mathbf{R}_8$. In the figure, the matrix $\mathbf{P}^{(2)}$ can be see to be two reverse shuffles $\mathbf{R}_4$.

6.2.3 Recursive Encoding

Encoding can be obtained by computing $\mathbf{c} = \mathbf{u}\mathbf{G}_N$, but this requires storage of the $\mathbf{G}_N$ matrix. Instead, a recursive encoding exploits the structure of the polar code. The encoder function is represented by:

$$\mathbf{c} = \text{Enc}(\mathbf{u}) \tag{6.30}$$

A subset of elements a to b from a vector $\mathbf{t}$ are represented as $\mathbf{t}_a^b = (t_a, t_{a+1}, \ldots, t_{b-1}, t_b)$.

Algorithm 6.1 $\mathbf{c} = \text{Enc}(\mathbf{u})$ Recursive Encoding of Polar Codes

Require: Information sequence $\mathbf{u}$ (including frozen bits) of length N.

Ensure: $\mathbf{c} = \mathbf{u}\mathbf{G}_N$

```
N' ← length of u
if N' = 2 then
    c_1 = u_1 + u_2
    c_2 = u_2
else
    r ← 1                                   ▷ r points to top half of reverse shuffle
    s ← N'/2 + 1                            ▷ s points to bottom half of reverse shuffle
    for i = 1, 2, 3, ..., N' do             ▷ Equivalent to t = u · (I_{N'/2} ⊗ F) · R_{N'}
        if i is odd then
            t_r = u_i + u_{i+1}
            r ← r + 1
        else if i is even then
            t_s = u_i
            s ← s + 1
        end if
    end for
    c_1^{N'/2}       = Enc(t_1^{N'/2})      ▷ Recursion
    c_{N'/2+1}^{N'} = Enc(t_{N'/2+1}^{N'})
end if
```

The function Enc performs the following steps with some $\mathbf{u}'$ with length N' as an input. If $N' = 2$ the encoding function is simply:

$$c_1 = u_1 + u_2 \tag{6.31}$$

$$c_2 = u_2 \tag{6.32}$$

and if $N' > 2$ then the encoding is performed recursively:

$$\begin{aligned} \mathbf{t} &= \mathbf{u} \cdot (\mathbf{I}_{N'/2} \otimes \mathbf{F}) \cdot \mathbf{R}_{N'} && \text{one step} \\ \mathbf{c}_1^{N'/2} &= \text{Enc}(\mathbf{t}_1^{N'/2}) && \text{encode upper bits} \\ \mathbf{c}_{N'/2+1}^{N'} &= \text{Enc}(\mathbf{t}_{N'/2+1}^{N'}) && \text{encode lower bits} \end{aligned}$$

which is equivalent to $\mathbf{c} = \mathbf{u}\mathbf{G}_N$. An algorithm implementing the recursive encoding is shown in Algorithm 6.1.

6.2.4 Reed-Muller Codes

Reed-Muller codes given in Section 2.4.5 can be seen as special type of polar code. The matrix $\mathbf{F}^{\otimes n}$ is related to Reed-Muller codes. Selecting certain rows

of $\mathbf{F}^{\otimes n}$ gives the generator matrix for Reed-Muller codes. Recall an order-r Reed-Muller code $\mathrm{RM}(r,n)$ has block length 2^n. The generator matrix of an $\mathrm{RM}(r,n)$ code are the rows of of $\mathbf{G}$ with weight greater than or equal to 2^{n-r}. There are $\sum_{i=0}^{r}\binom{n}{i}$ such rows, so this is the dimension K of the code.

6.3 Successive Cancelation Decoding

6.3.1 Principle of Successive Cancelation Decoding

A codeword from a polar code $\mathbf{c}=(c_1,c_2,\ldots,c_N)$ is transmitted over a binary-input memoryless channel with input c and output y. Usually, the channel is denoted $\Pr(y|c)$, but in the context of polar codes the convention is to instead write $W(y|c)$; $\Pr(\cdot)$ and $W(\cdot)$ mean the same thing. Thus $\mathbf{c}$ is transmitted over N uses of the channel $W(y|c)$ and $\mathbf{y}$ is received.

The principle of successive cancelation decoding is introduced. In successive cancelation decoding of polar codes, the decoder makes a hard decision for bit u_i using all the previous estimates $\widehat{u}_1,\widehat{u}_2,\ldots,\widehat{u}_{i-1}$ as well as the entire received sequence $\mathbf{y}=y_1,\ldots,y_N$. Decoding begins by making an estimate $\widehat{u}_1\in\{0,1\}$ using $\mathbf{y}$. Then the decoder makes an estimate $\widehat{u}_2\in\{0,1\}$ using $\widehat{u}_1$ and $\mathbf{y}$. Successive cancelation decoder proceeds in this way until it estimates $\widehat{u}_N$ using $\widehat{u}_1,\ldots,\widehat{u}_{N-1}$ and $\mathbf{y}$. Frozen bits are known and do not need to be estimated.

Suppose $W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i)$ is known (or equivalently, $\Pr(\mathbf{y},\mathbf{u}_1^{i-1}|u_i)$ is known). The hard decision in position i is $\widehat{u}_i=0$ if:

$$W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=0)>W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=1), \tag{6.33}$$

and otherwise the hard decision is $\widehat{u}_i=1$. The decision is equivalent to:

$$\widehat{u}_i=\begin{cases}0 & W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=0)>W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=1)\\ 1 & \text{otherwise}\end{cases} \tag{6.34}$$

In order to make hard decision, the two needed probabilities (or channels) are $W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=0)$ and $W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i=1)$. These are found by *channel splitting*.

6.3.2 Channel Splitting

The key idea in polar codes is to transform N copies of the channel W are transformed to N synthetic channels, some of which are better than than W and some of which are worse than W. This subsection shows how to find the synthetic channel $W(\mathbf{y},\mathbf{u}_1^{i-1}|u_i)$ from the physical channel $W(y|c)$. Later we show how some of these channels are good, and some are bad.

It is possible to synthesize a vector channel $W_N(\mathbf{y}|\mathbf{u})$ from N copies of the single channel $W(y|x)$ using the recursive structure of polar codes. In this section, this vector channel is split into N synthetic channels:

$$W_N^{(i)}\big(\underbrace{\mathbf{y},\mathbf{u}_1^{i-1}}_{\text{output}}\big|\underbrace{u_i}_{\text{input}}\big) \tag{6.35}$$

for $i = 1, 2, \ldots, N$. As with any channel described by a conditional probability distribution, variables to the left are "outputs" and the variable on the right are "inputs". At step i of successive cancelation decoding, the decoder uses $u_1, u_2, \ldots, u_{i-1}$ and $\mathbf{y}$, to produce an estimate $\widehat{u}_i$. These are called synthetic channels because they do not correspond to physical channels, but indeed are conditional probability distributions.

For example with $N = 4$, the goal is to find the synthetic channels:

$$\begin{aligned} W_4^{(1)} &= W_4^{(--)}(y_1^4|u_1) \\ W_4^{(2)} &= W_4^{(-+)}(y_1^4, u_1|u_2) \\ W_4^{(3)} &= W_4^{(+-)}(y_1^4, u_1^2|u_3) \\ W_4^{(4)} &= W_4^{(++)}(y_1^4, u_1^3|u_4) \end{aligned}$$

Here, $+, -$ is used for a binary representation of integers. In what follows, both representations will be used.

If $W_N^{(1)}, W_N^{(2)}, \ldots, W_N^{(N)}$ can be obtained, then these can be used for decoding. The following proposition shows how to compute the synthetic channels recursively.

Proposition 6.2 (Arikan 2008, Proposition 3)**.** For any $n \geq 0$, $N = 2^n$ and $1 \leq i \leq N$,

$$W_{2N}^{(2i-1)}(y_1^{2N}, u_1^{2i-2}|u_{2i-1}) = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_1^N, u_{1,\text{odd}}^{2i-2} + u_{1,\text{even}}^{2i-2}|u_{2i-1} + u_{2i}) W_N^{(i)}(y_{N+1}^{2N}, u_{1,\text{even}}^{2i-2}|u_{2i}) \tag{6.36}$$

$$W_{2N}^{(2i)}(y_1^{2N}, u_1^{2i-1}|u_{2i}) = \frac{1}{2} W_N^{(i)}(y_1^N, u_{1,\text{odd}}^{2i-2} + u_{1,\text{even}}^{2i-2}|u_{2i-1} + u_{2i}) W_N^{(i)}(y_{N+1}^{2N}, u_{1,\text{even}}^{2i-2}|u_{2i}) \tag{6.37}$$

The two colored terms are the same, showing the similarity between the two equations. The proposition gives a recursive means to calculate the synthetic channels. The recursion proceeds until $W(y|u)$ is reached — this is the physical channel distribution, which is known.

6.3.3 Channel Splitting for the BEC

This subsection considers the binary erasure channel, for which computation of the synthetic channels is straightforward.

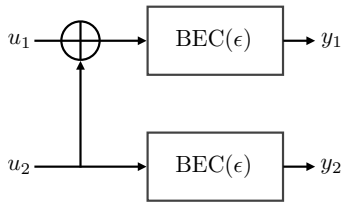
With $N = 1$, consider a BEC with input u, output y and erasure probability ϵ. We can write a decoding table for estimating the input $\widehat{u}$ given the output y:

y	$\widehat{u}$
u	u
?	?

For the case of $N = 2$, consider the two synthetic channels $W_2^{(-)}$ and $W_2^{(+)}$,

$$W_2^{(-)} = W(y_1, y_2|u_1) \tag{6.38}$$
$$W_2^{(+)} = W(y_1, y_2, u_1|u_2) \tag{6.39}$$

Above, $W_2^{(-)}$ is the same as $W_2^{(1)}$ and $W_2^{(+)}$ is the same as $W_2^{(2)}$. In what follows, both representations will be used.



First consider the upper branch or "-" branch, decoding u_1 using y_1, y_2. The decoding table is:

y_1	y_2	$\widehat{u}_1$
$u_1 + u_2$	u_2	u_1
?	u_2	?
$u_1 + u_2$	?	?
?	?	?

On the BEC with erasure probability ϵ, the probability of erasure $\Pr(\widehat{u}_1 =?)$ is $1 - (1 - \epsilon)(1 - \epsilon) = 2\epsilon - \epsilon^2$.

For decoding the lower branch or "+" branch, decode u_2 using u_1, y_1, y_2. Since u_1 is always available, the decoding table is:

y_1	y_2	u_1	$\widehat{u}_2$
$u_1 + u_2$	u_2	u_1	u_2
?	u_2	u_1	u_2
$u_1 + u_2$	?	u_1	u_2
?	?	u_1	?

On the BEC with erasure probability ϵ, the probability that $\widehat{u}_2$ is erased is ϵ^2.

The probability of erasure for $N = 2$ depends on the branch, and is expressed using the function $T^{(s)}$:

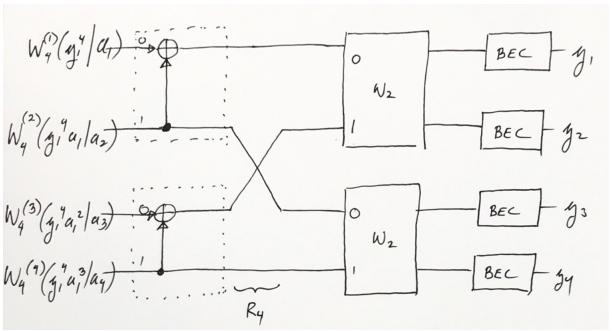
$$T^{(s)}(\epsilon) = \begin{cases} 2\epsilon - \epsilon^2 & s = - \\ \epsilon^2 & s = + \end{cases} \tag{6.40}$$

Now for $N = 4$, we can apply the polar transform of the proposition but it

is helpful to write recursively, also as shown in the figure below:

$W_4^{(--)}(y_1^4\|u_1)$	Obtained by the $-$ transformation on $W_2^{(-)}$
$W_4^{(-+)}(y_1^4, u_1\|u_2)$	Obtained by the $+$ transformation on $W_2^{(-)}$
$W_4^{(+-)}(y_1^4, u_1^2\|u_3)$	Obtained by the $-$ transformation on $W_2^{(+)}$
$W_4^{(++)}(y_1^4, u_1^3\|u_4)$	Obtained by the $+$ transformation on $W_2^{(+)}$

The four channels are the same as $W_4^{(1)}, W_4^{(2)}, W_4^{(3)}, W_4^{(4)}$. The $+/-$ is a binary string with the LSB on the left.



Now it is possible to find the probability of erasure for $N = 4$ using the lower level. If p is the probability of erasure at level W_2, then the probability of erasure for $N = 4$ is:

$$T^{(s)}(p) = \begin{cases} 2p - p^2 & s = - \\ p^2 & s = + \end{cases} \tag{6.41}$$

This is valid because the probability of erasure for both copies of W_2 are the same. For $N = 8, 16$, etc. on the BEC, the recursion is the same, where p represents the probability of erasure one level lower.

Let $\mathbf{s} = [s_1, s_2, \ldots, s_n]$ be the binary representation of bit position i (indexed from one), so that $1 = ----$, $2 = +---$, $3 = -+--$, $\ldots$, $16 = ++++$. The probability of erasure for position s on BEC(ϵ) is denoted by $S^{(\mathbf{s})}(\epsilon)$ and can be found recursively as:

$$S^{(\mathbf{s})}(\epsilon) = T^{(s_n)}\left(T^{(s_{n-1})}\left(\cdots\ T^{(s_1)}\left(\epsilon\right)\right)\right). \tag{6.42}$$

For the decoder to succeed, all bits in $\mathcal{A}$ must be decoded correctly. The probability of decoding error under successive cancellation decoding, or the word error rate (WER) can be found analytically as one minus the probability that all synthetic channels decode successfully:

$$\text{WER} = 1 - \prod_{i \in \mathcal{A}} (1 - S^{(i)}(\epsilon)) \tag{6.43}$$

Example 6.6. Consider an $N = 8$ polar code on the BEC with $\epsilon = 0.1$.

Evaluate $S^{(1)}(\epsilon)$ and $S^{(7)}(\epsilon)$. For $S^{(1)}(\epsilon)$, evaluate numerically at each recursion step because the analytic form is not simple:

$$\begin{aligned} S^{(1)} = S^{(---)} &= T^{(-)}\left(T^{(-)}\left(T^{(-)}\left(\epsilon\right)\right)\right) \\ &= T^{(-)}\left(T^{(-)}\left(2\epsilon - \epsilon^2\right)\right) \\ &= T^{(-)}\left(T^{(-)}\left(0.19\right)\right) \\ &= T^{(-)}\left(2 \cdot 0.19 - 0.19^2\right) \\ &= T^{(-)}\left(0.3439\right) \\ &= 0.5695 \end{aligned}$$

For $S^{(7)}(\epsilon)$, the analytical form is simple:

$$\begin{aligned} S^{(7)} = S^{(-++)} &= T^{(-)}\left(T^{(+)}\left(T^{(+)}\left(\epsilon\right)\right)\right) \\ &= T^{(-)}\left(T^{(+)}\left(\epsilon^2\right)\right) \\ &= T^{(-)}\left(\epsilon^4\right) \\ &= 2\epsilon^4 - \epsilon^8 \\ &= 0.0002 \end{aligned}$$

Note that bit position 1 has a higher probability of erasure than ϵ, while bit position 7 has a lower probability of erasure.

6.4 Polarization Phenomenon and Code Design

6.4.1 Polarization

The probability of erasure for the synthetic channels $S^{(i)}$ can be found for $i = 1, 2, \ldots, N$. These can be sorted from smallest to largest, which is shown in Fig. 6.4 for $N = 8, 16, 256, 16384$. As N increases, clear pattern emerges — some bit positions have probability of erasure close to 0, while other bit positions have probability of erasure close to 1. For moderate values such as $N = 256$, there is a transition region as well. But as N increases, a *polarization* effect occurs – the transition region becomes sharp, and most bits are either highly reliable (probability of erasure close to 0) or are highly unreliable (probability of erasure close to 1). This interesting phenomenon gives polar codes their names.

6.4.2 Polar Code Design

Polar code design means selecting the set of information bits $\mathcal{A}$, for a given block length N and dimension K.

This suggests a code design technique — to design a polar code, use the K most reliable synthetic channels to transmit information. For the remaining $N - K$ channels, freeze the information bits to 0.

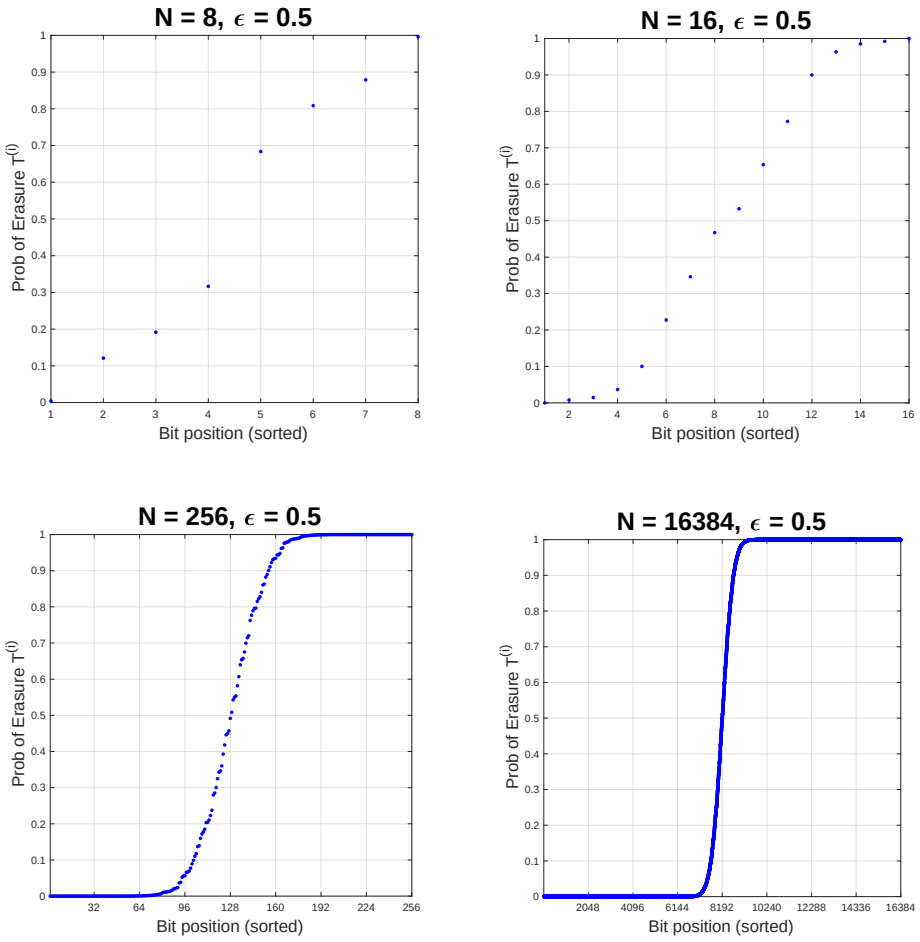


Figure 6.4: Probability of erasure for each bit position. The bit positions have been sorted according to increasing probability of erasure. For $N = 8, 16, 256, 16384$.

Example 6.7. Design a polar code with $N = 8$ and $K = 5$, continuing Example 6.6. The probability of erasure for all $N = 8$ synthetic channels are:

$$\begin{aligned} S^{(1)} &= 0.569533 & S^{(5)} &= 0.039404 \\ S^{(2)} &= 0.118267 & S^{(6)} &= 0.00039601 \\ S^{(3)} &= 0.0708968 & S^{(7)} &= 0.0002 \\ S^{(4)} &= 0.00130321 & S^{(8)} &= 10^{-8} \end{aligned}$$

The $K = 5$ best channels are 4 to 8, so we choose the information set to be $\mathcal{A} = \{4, 5, 6, 7, 8\}$.

6.5 Non-Recursive Successive Cancelation Decoding

Using the recursive decoder can be inefficient in computers because function calls in software often have a large overhead. There is also a lot of redundancy in those function calls, when decoding all the bits of a codeword. Use the block diagram for decoding. Note that if the SC decoder makes an error in decoding position i, this error will propagate forward, affecting following decisions.

For each information bit u_i for $i \in \mathcal{A}$:

1. Forward stage: moving left-to-right, compute all possible bits. Use all frozen bits and hard decisions so far.
2. Backward stage: moving right-to-left, compute all possible log values
3. For bit u_i, make a hard decision using its log value:

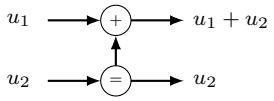
$$\widehat{u}_i = \begin{cases} 0 & \beta_i^{(1)} \geq 0 \\ 1 & \beta_i^{(1)} < 1 \end{cases} \tag{6.44}$$

6.5.1 Decoder Element

The decoding algorithm uses a decoding element, and first "local" operations using this decoding algorithm are discussed. There are two stages: (1) forward computation of hard decisions (2) backward computation of soft values. Each stage has an upper branch and a lower branch computation.

The forward stage is a hard-input, hard-output. Inputs on the left are used to compute outputs on the right.

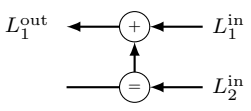
Forward Stage For the upper branch, if $u_1, u_2 \in \{0, 1\}$ are available at the left-hand side input, then the right-hand side upper output is $u_1 + u_2$. If both u_1 and u_2 are not available, then there is no lower output. For the lower branch, if $u_2 \in \{0, 1\}$ is available at the input, then output is u_2. If u_2 is not available, then there is no lower output.



The backward stage is a soft-input, soft-output. Inputs on the right are used to compute outputs on the left. Consider the basic element of the polar code which represents $(c_1, c_2) = (u_1 + u_2, u_2)$. There are two inputs on the right, and two outputs on the left. In addition, when calculating the lower branch there may be a hard decision on u_1 available. The soft messages are in the log domain.

$$L = \log \frac{\Pr[x = 0]}{\Pr[x = 1]} \tag{6.45}$$

Backward Stage, Upper Branch The decoder element has inputs $L_1^{\text{in}}, L_2^{\text{in}}$, and upper branch output L_1^{out}:

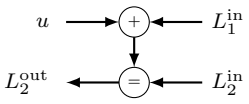


This output is computed as:

$$L_1^{\text{out}} = 2\tanh^{-1}\left(\tanh(\frac{L_1^{\text{in}}}{2})\tanh(\frac{L_2^{\text{in}}}{2})\right). \tag{6.46}$$

If $L_1^{\text{in}}, L_2^{\text{in}}$, are not available, then the output cannot be computed.

Backward Stage, Lower Branch The decoder element has inputs $L_1^{\text{in}}, L_2^{\text{in}}$, and a hard decision input from the left $u_1 \in \{0, 1\}$. The lower branch output is L_2^{out}:



This output is computed as:

$$L_2^{\text{out}} = \begin{cases} L_1^{\text{in}} + L_2^{\text{in}} & \widehat{u}_1 = 0 \\ -L_1^{\text{in}} + L_2^{\text{in}} & \widehat{u}_1 = 1 \end{cases} \tag{6.47}$$

which can also be written as $L_2^{\text{out}} = (-1)^{u_1} L_1^{\text{in}} + L_2^{\text{in}}$. If $L_1^{\text{in}}, L_2^{\text{in}}, u_1$, are not available, then the output cannot be computed.

6.5.2 Example

Consider an $N = 8$ polar code with information bits $\mathcal{A} = \{4, 6, 7, 8\}$. The received log-domain messages are:

$$[-2.4, 9.2, -2.4, -1.6, -3.52, 0.28, 2.8, -3.04]; \tag{6.48}$$

Examples of decoding for each bit are illustrated on the following figures.

6.6 References

1. E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, Vol. 55, No. 7, July 2009

2. Alexios Balatsoukas-Stimming, "Efficient Decoding of Polar Codes - Algorithms and Implementations," 2020 European School on Information Theory, video available at https://vimeo.com/showcase/7823297/video/482236401. Use password #StayHomeStaySafe

3. H. D. Pfister, "A brief introduction to polar codes," Lecture notes, Duke University, October 8, 2017. Available at http://pfister.ee.duke.edu/courses/ecen655/polar.pdf

4. S. Aizaz A. Shah, "Polar codes," Hamburg University of Technology.

5. Successive Cancellation(SC) Decoder for Polar Codes: Illustration of its Building Blocks with N=2,4, YouTube https://www.youtube.com/watch?v=wK2KI2LtdQI

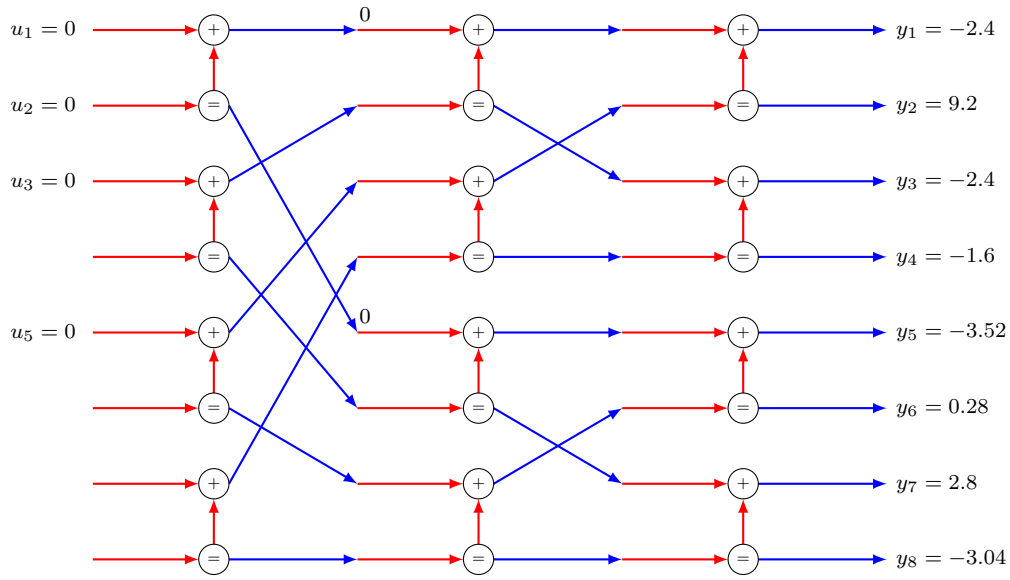


Figure 6.5: Decoding $N = 8$ polar code, information bit position 4, forward part only.

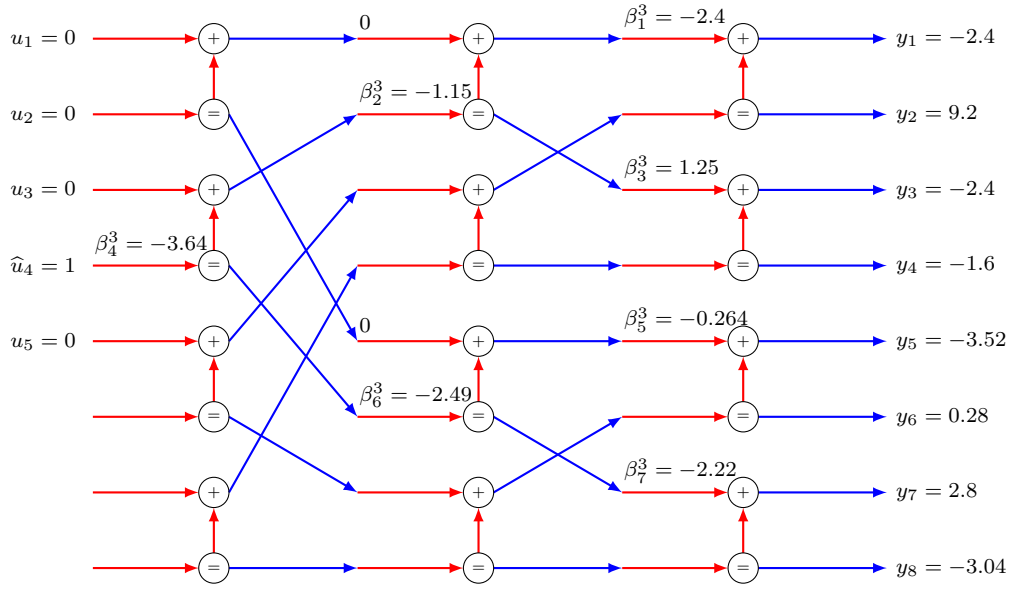


Figure 6.6: Decoding $N = 8$ polar code, information bit position 4, obtaining $\widehat{u}_4$.

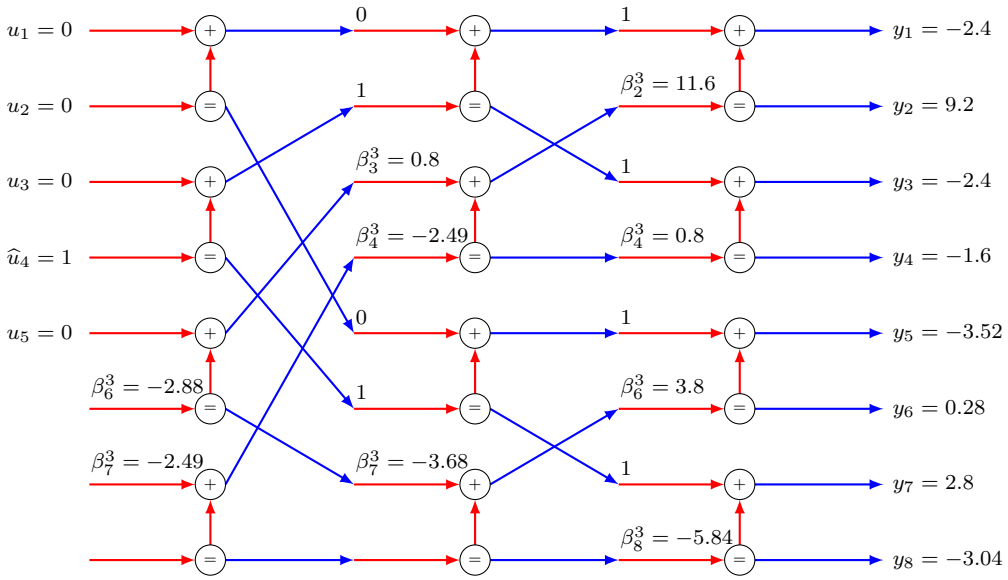


Figure 6.7: Decoding $N = 8$ polar code, information bit position 6, obtaining $\widehat{u}_6$.

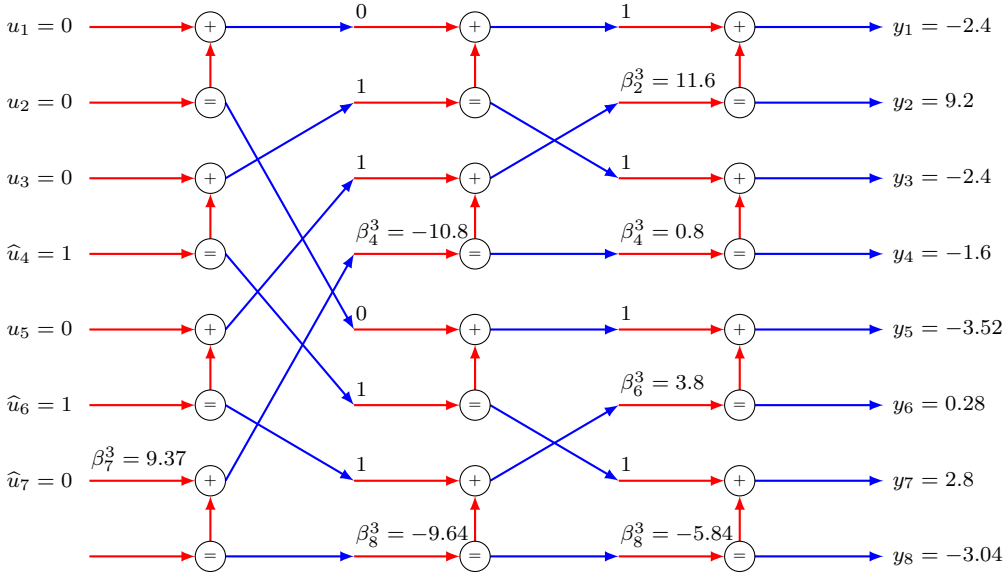


Figure 6.8: Decoding $N = 8$ polar code, information bit position 7, obtaining $\widehat{u}_7$.

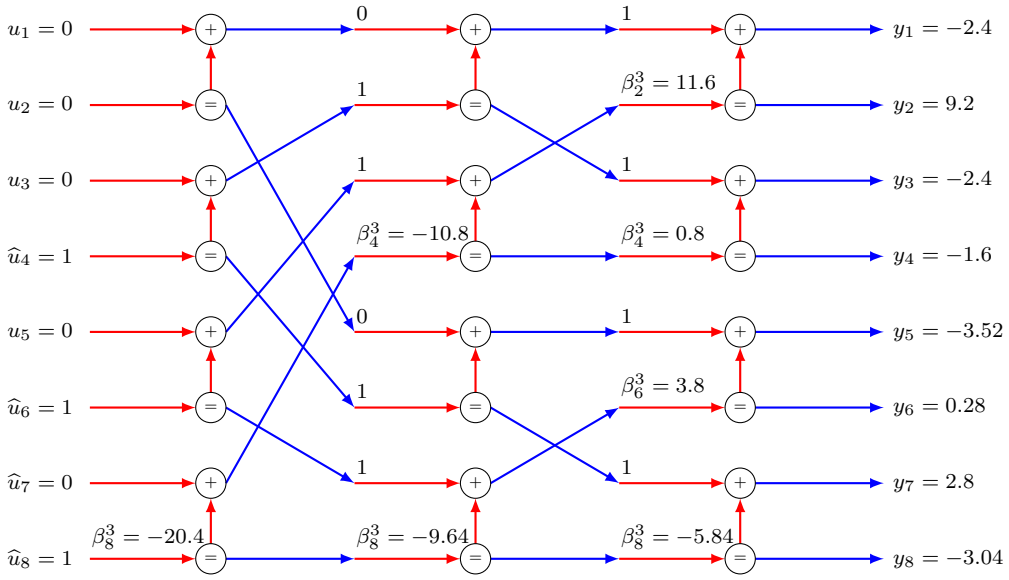


Figure 6.9: Decoding $N = 8$ polar code, information bit position 8, finally obtaining $\widehat{u}_8$.

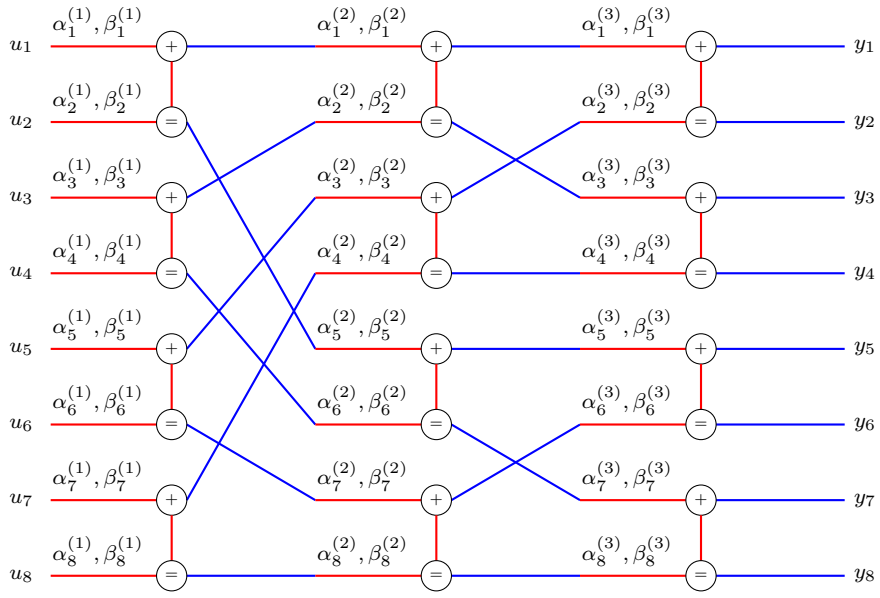


Figure 6.10: Definition of messages for decoding $N = 8$ polar code.

6. A. Montanari, "Polar codes," Lecture Notes, Stanford University, May 2018, https://web.stanford.edu/class/ee388/HOMEWORK2018/lecture-9-10-11.pdf

7. Kai Niu, Kai Chen, Jiaru Lin and Q. T. Zhang, "Polar Codes: Primary Concepts and Practical Decoding Algorithms," *IEEE Communications Magazine*, July 2014. IEEE Xplore Link

6.7 Exercises

6.1 Let $\mathbf{F}^{\otimes n}$ be the $2^n \times 2^n$ Hadamard matrix. Over the field $\mathbb{F}_2$, prove that:

(a) $\left(\mathbf{F}^{\otimes n}\right)^2 = \mathbf{I}_{2^n}$, the identity matrix.
(b) $\left(\mathbf{F}^{\otimes n}\right)^{-1} = \mathbf{F}^{\otimes n}$. The Hadamard matrix is its own inverse.

6.2 Find the $N = 16$ bit-reversal shuffle $\mathbf{B}_{16}$. Write your answer in the form of (6.14).

6.3 Show the $N = 4$ matrix decomposition for $\mathbf{G}_4$.

(a) What is the relationship between $\mathbf{B}_4$ and $\mathbf{R}_4$?
(b) The matrix decomposition for $N = 4$ is $\mathbf{G}_4 = (\mathbf{I}_2 \otimes \mathbf{F}) \cdot \mathbf{R}_4 \cdot (\mathbf{I}_2 \otimes \mathbf{F})$. Using the perfect shuffle to commute the Kronecker product, equation (10.2) , show that:

$$\mathbf{G}_4 = \mathbf{B}_4 \mathbf{F}^{\otimes 2} \tag{6.50}$$

6.4 (a) Find the probabilities of erasure decoding $S^{(i)}$, $i = 1, 2, \ldots, 8$ for an $N = 8$ polar code on the BEC with $\epsilon = 0.4$. (b) Give the 3×8 generator matrix for an $N = 8$, $K = 3$ polar code for use on this channel. (c) What is the word error rate (WER)?

Chapter 7

Convolutional Codes

A convolutional code is an error-correcting code that generates codeword by applying a polynomial function to the uncoded information. This application is performed by the convolution of the encoder with the information. This convolutional codes enables decoding using a *trellis*. Maximum-likelihood decoding can be performed with reasonable complexity.

7.1 Convolutional Codes

7.1.1 Definition and Encoding

Consider an input sequence $u_1, u_2, \ldots, u_k$. Use delay-operator notation, so this can be represented as a polynomial:

$$u_1 + u_2 D + \cdots + u_k D^{k-1} \tag{7.1}$$

A convolutional code has one input $u(D)$ and f outputs, $c^{(1)}(D), \ldots, c^{(f)}(D)$. For output i, there is a generator polynomial $g^{(i)}$

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)} + \cdots + g_\nu^{(i)} D^\nu \tag{7.2}$$

for $i = 1, 2, \ldots, f$. Each output is generated by convolution by a generator polynomial:

$$c^{(i)}(D) = u(D) * g^{(i)}(D), \tag{7.3}$$

where the $*$ operation is given by:

$$c_i = \sum_{j=0}^{\nu} u \cdot g \tag{7.4}$$

We like to write the input as a codeword by grouping the f outputs together. For example if $f = 3$:

$$\mathbf{c} = [c_1^{(1)} c_1^{(2)} c_1^{(3)}, c_2^{(1)} c_2^{(2)} c_2^{(3)}, \ldots, c_n^{(1)} c_n^{(2)} c_n^{(3)}] \tag{7.5}$$

So with that, we can write the generator matrix in matrix form:

Example 7.1. Consider the $f = 2$ code with:

$$g^{(1)}(D) = D^2 + D + 1 \tag{7.6}$$
$$g^{(2)}(D) = D^2 + 1 \tag{7.7}$$

Then the generator matrix looks like:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.8}$$

We describe the construction of convolutional code lattices based on rate $1/f$ convolutional codes and Construction A. The permutations on the generator matrix of convolutional codes that produce a canonical form is described.

Convolutional code of rate $1/f$ are described. Consider a rate $1/f$ non-systematic forward convolutional code $\mathcal{C}$. The encoder has input consisting of one information sequence and a codeword consisting of f output sequences. The constraint length ν is defined as the number of shift register delay elements. The encoder has one shift register of ν delay elements. The generator polynomials are:

$$\begin{aligned} \mathbf{g}^{(0)}(D) &= g_0^{(0)} + g_1^{(0)} D + \cdots + g_\nu^{(0)} D^\nu \\ \mathbf{g}^{(1)}(D) &= g_0^{(1)} + g_1^{(1)} D + \cdots + g_\nu^{(1)} D^\nu \\ &\vdots \\ \mathbf{g}^{(f-1)}(D) &= g_0^{(f-1)} + g_1^{(f-1)} D + \cdots + g_\nu^{(f-1)} D^\nu, \end{aligned}$$

where D is the delay operator. For convenience, the generator polynomials can be transferred to binary sequences, and then be represented by octal sequences. The binary sequences correspond to the coefficients of generator polynomials in an ascending order.

The generator matrix of a rate $1/f$ convolutional code $\mathbf{G}(D) = [\mathbf{g}^{(0)}(D)\mathbf{g}^{(1)}(D) \dots \mathbf{g}^{(f-1)}(D)]^{\mathrm{t}}$ is equivalent to $\mathbf{G}(D) = \mathbf{G}_0 + \mathbf{G}_1 D + \cdots + \mathbf{G}_\nu D^\nu$, where $\mathbf{G}_i = [g_i^{(0)} g_i^{(1)} \dots g_i^{(f-1)}]^{\mathrm{t}}$,

for $i = 0, 1, \ldots, \nu$. The convolutional code generator matrix can be written as:

$$\mathbf{G}' = \begin{bmatrix} \mathbf{G}_0 & & \\ \mathbf{G}_1 & \mathbf{G}_0 & \\ \vdots & \vdots & \ddots \\ \mathbf{G}_\nu & \mathbf{G}_{\nu-1} & \\ & \mathbf{G}_\nu & \ddots \\ & & \ddots \end{bmatrix}, \tag{7.9}$$

where each submatrix $\mathbf{G}_i$ consists of f rows and 1 column. If k is the length of the information vector, $\mathcal{C}$ can be terminated to a block length of $n = f(k + \nu)$. That is, the generator matrix $\mathbf{G}$ is an $n \times k$ matrix.

7.1.2 Decoding Convolutional Codes