Lecture Notes for

SSY125 Digital Communications

Alexandre Graell i Amat

Chalmers University of Technology
Department of Electrical Engineering
alexandre.graell@chalmers.se
https://sites.google.com/site/agraellamat/

November 5, 2019

The following acronyms are used in this manuscript.

Acronym	**Meaning**
APP	A posteriori probability
AWGN	Additive white Gaussian noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit error rate
BP	Belief propagation
BPSK	Binary phase shift keying
BSC	Binary symmetric channel
CN	Check node
ECC	Error correcting code
LDPC	Low-density parity-check code
LLR	Log-likelihood ratio
MAP	Maxium a posteriori
ML	Maximum likelihood
PDF	Probability density function
PMF	Probability mass function
QAM	Quadrature amplitude modulation
RSC	Recursive systematic convolutional
SER	Symbol error rate
SISO	Soft-input soft-output
SNR	Signal-to-noise ratio
SPC	Single parity-check code
VN	Variable node

Contents

Chapter 1

Introduction

The fundamental problem of communication is that of sending information from one point to another (either in space or time) efficiently and reliably. Examples of communication are the data transmission between an Earth station and a space probe through the atmosphere and free space, the write and read of information in a flash memory, as well as communication between servers in a data center through optical interconnects. The physical medium, e.g., the free space or the optical fiber, is prone to noise, distortion, and other impairments. As a consequence, the transmitted (digital) message may be received with errors.

The foundation of the information age dates back to Claude E. Shannon and his 1948 landmark paper "A mathematical theory of communication". Shannon provided a mathematical framework for communication, giving answer to two fundamental questions: What is information and how can we measure it? What are the limits of storage and reliable transmission and how can we achieve them? To give answer to these questions, Shannon provided a mathematical notion of information where the sender and the receiver, as well as the meaning of the message, are irrelevant. The beauty of this abstract notion of information is that it allows information to be measured and manipulated. Hence, it is at the basis of the information era.

Shannon showed that information can efficiently be represented with a sequence of bits, the fundamental unit of information, and is closely associated with uncertainty: the information content of a (data) source is related to the amount of uncertainty on its outcome. He called it entropy. He also realized that content can be somehow compressed (without loss of information) to allow for faster transmission and showed that the entropy of a source is the fundamental limit for data compression. He also showed that, for most transmission channels, there is a fundamental limit at which information can be transmitted reliably (i.e., with vanishingly small error probability), the *channel capacity*. Furthermore, he also showed how this limit can be achieved: by the use of coding.

Shannon's fundamental contribution marks the birth of the fields of information theory, source compression, and coding theory. While information theory aims at determining the fundamental limits of communication, source compression and coding theory aim at finding codes that achieve (or approach) these limits in practice.

Information theory, source compression, and coding theory are vast fields that are impossible to cover in a (8-week) master's course. This is certainly not the aspiration of this course. The aim of this course is simply to provide a brief and accessible introduction to the main concepts underlying these fields, thus giving answers to the two fundamental questions above.

The remainder of this manuscript is organized as follows. In Chapter 2, we give a mathematical definition of information and of the information content of a source, the entropy. Chapter 3 is devoted to source compression, i.e., the efficient representation of the information content of a source. We show that the fundamental limit of (lossless) source compression is given by the entropy and we provide an algorithm for optimal (lossless) compression. In Chapter 4, we discuss communication over a noisy channel and the fundamental limit at which information can be transmitted reliably, the channel capacity. In Chapter 5, we particularize the capacity for the important case of transmission over the additive white Gaussian noise (AWGN) channel. In Chapter 6, we review the optimum decision rule and uncoded transmission, which will serve us as a benchmark to compare the performance of coding schemes. In Chapter 7, we introduce the main concepts underlying coding. Chapters 8 and 9 review two of the most important classes of codes, block codes and convolutional codes, respectively. Finally, in Chapters 10 and 11 we introduce two of the main classes of modern codes that are currently used in a myriad of communication standards and systems, namely turbo codes and low-density parity-check codes.

Notation: The following notation is used throughout this manuscript. We use boldface letters to denote vectors and matrices, e.g., $\boldsymbol{x}$ and $\boldsymbol{X}$, respectively, and calligraphic letters to denote sets, e.g., $\mathcal{X}$. Unless otherwise stated, vectors are defined as row vectors. We use $\mathbf{0}_a$ and $\mathbf{0}_{a\times b}$ to denote the all-zero vector of length a and the all-zero matrix of dimensions $a \times b$, $\mathbf{1}_a$ to denote the all-one vector of length a, and $\boldsymbol{I}_{a\times b}$ to denote the identity matrix of dimensions $a \times b$. The cardinality of a set $\mathcal{X}$ is denoted as $|\mathcal{X}|$. Probability is denoted by $\Pr(\cdot)$. The probability mass function (PMF) and the probability density function (PDF) of a random variable (RV) X are denoted by $P_X(x)$ and $p_X(x)$, respectively, as a shorthand for $P_X(X = x)$ and $p_X(X = x)$. For notational simplicity, sometimes we will simply write $P(x)$ and $p(x)$. Also, we write $p_{XY}(x, y)$ and $p_{Y|X}(y|x)$ to denote the joint PDF of two RVs X and Y and the conditional PDF of Y conditioned on $X = x$, respectively, and we use similar notation for the joint and conditional PMFs. Expectation is denoted by $\mathbb{E}$ and the expectation of a function $g(X)$ with respect to p_X is denoted as $\mathbb{E}_{p_X}[g(x)]$, or, in short $\mathbb{E}_X[g(x)]$, i.e., $\mathbb{E}_X[g(x)] = \int g(x)p_X(x)dx$. Vector and matrix transpose are denoted by $(\cdot)^{\mathsf{T}}$, and $\|\boldsymbol{x}\|$ denotes the norm of $\boldsymbol{x}$.

Chapter 2

A Measure of Information

THE AIM of any communication system is to **send information efficiently and reliably** from one place to another or from one time to another (think about the music stored in your smartphone or the data stored in a hard-disk drive. The data is stored *now*, and will be retrieved at some time in the future. In this case, we talk about **storage**, which can be seen as a communication problem where information is transmitted from the *present* to the *future*, and involves the write and read of the information).

Everyone has more or less a notion of what information means. However, **how do we measure information?**, and, **how can we mathematically describe an information source?** In this chapter we will address these fundamental questions.

In this course we will consider only **discrete information sources**, i.e., the information source generates a sequence of symbols that take values on a finite alphabet $\mathcal{X}$. This is obviously true for a digital source. An analog source, on the other hand, can always be digitalized (this will not be covered in this course, but the student is supposed to be familiar with analog-to-digital conversion). Examples of discrete alphabets are the roman alphabet, the characters of a keyboard, or the symbols in a sheet music.

It is important to realize that the output of any information source is random, i.e., we don't know a priori which message will be transmitted (otherwise transmission wouldn't be needed!). In other words, **the output of an information source is a sequence of random variables** $X^{(1)}X^{(2)}X^{(3)}\ldots$ at times $i = 1, 2, 3, \ldots$. Therefore, a source can be defined by a discrete-time stochastic process. Generally speaking, a discrete information source produces a sequence of random variables $X^{(1)}X^{(2)}X^{(3)}\ldots$ (random symbols forming a message), where each $X^{(i)}$ takes values on a finite alphabet $\mathcal{X} = \{x_1, x_2 \ldots, x_M\}$, of cardinality $|\mathcal{X}| = M$, and PMF $P_X(x)$, such that $\Pr(X = x_i) = p_i$ and

$$p_i \geq 0 \ \forall i, \quad \text{and} \quad \sum_{x_i \in \mathcal{X}} p_i = 1. \tag{2.1}$$

The source is completely described by $\mathcal{X}$ and $P_X(x)$. In this course, we will restrict ourselves to **discrete memoryless sources**, i.e., sources where the random variables $X^{(i)}$ are independent of each other.

Now that we have defined an *information source*, we need to give a notion of information. An intuitive notion of information refers to any new knowledge about an event. The information source outputs messages which are of interest to the receiver, who does not know the messages a priori. The role of the communications system is then to make sure that the information is conveyed efficiently and reliably to the receiver.

We have seen that, in mathematical terms, the output of a source is a random variable X. How can we measure the information content of a particular outcome $X = x_i$ from the source? And more generally, how much information does a source contain? Let's try to derive a measure of information such that some intuitive properties are satisfied. Consider the discrete information source above and, without loss of generality, assume $p_1 \geq p_2 \geq \ldots \geq p_M$. Which message brings more information, the most likely one, x_1, or the most unlikely, x_M? Intuitively, revealing x_M will provide more information. In other words, **the information associated to a particular message x is related to its uncertainty**: The more uncertain the message is, the more information it conveys. Let's clarify this with a very simple example.

Example 2.1 Consider the following two sentences (related to the weather in November in Gothenburg).

- Tomorrow it will rain.
- Tomorrow there will be 30°C.

The first sentence brings little information, since it is quite expected that it will be raining in Gothenburg in November. However, the second sentence seems to contain much more information, since having 30°C in November is very unexpected.

From the discussion above we can now establish that a good measure of the information associated to a symbol x, denoted by $\mathsf{i}(x)$, must be a **decreasing function of the probability of the symbol**, i.e.,

$$\mathsf{i}(x_i) > \mathsf{i}(x_j) \quad \text{if} \quad p_i < p_j. \tag{2.2}$$

A second intuitive property of the information is that a slight change in the probability of a symbol should only cause a slight change in the information that it conveys. In other words, a measure of information should be a **continuous function of the probability**. Furthermore, we would expect that **two equiprobable symbols carry the same amount of information**, i.e., a good measure of the information should not depend on the symbol itself, but only on its probability.

Let's consider now another example.

Example 2.2 Imagine learning the value of two independent random variables. For example, consider that we roll a dice 2 times and communicate the outcome to a friend. Intuitively, we want any measure of information to have the property of addition, i.e., for independent random variables X and Y the information we gain when we learn X and Y should equal the sum of the information gained if we learn X and Y separately. In our example, the amount of information that we convey when we roll a dice twice is two times the amount of information that we convey when rolling the dice only once.

From this example we can infer another intuitive property that a measure of information should satisfy: The information should be **additive**; if two events (symbols) are independent, the information conveyed is the sum of the information conveyed by revealing each of the events separately.

In summary, the content of information revealed by a symbol x_i of probability p_i should have the following three properties.

P1 The information contained by a symbol x_i should **depend exclusively on the probability** $P(x_i)$, and not on the value x_i.

P2 The information should be a **continuous, decreasing function of the probability**,

$$\mathsf{i}(x_i) > \mathsf{i}(x_j) \quad \text{if} \quad p_i < p_j, \tag{2.3}$$

i.e., the lower the probability of the symbol, the larger the information should be. Obviously, if the probability of a symbol is 1, the corresponding information should be 0.

- P3 The information associated to the transmission of two independent symbols should be equal to the information that the two symbols convey separately, i.e., **the sum of information**: If the events $X = x_i$ and $X = x_j$ are independent, i.e., $P(x_i, x_j) = P(x_i)P(x_j)$, then

$$\mathsf{i}(x_i, x_j) = \mathsf{i}(x_i) + \mathsf{i}(x_j). \tag{2.4}$$

2.1 Shannon's Information Measure

The logarithmic function captures the three properties above. We therefore define the following measure of information.

Definition 2.1 The Shannon information content of the outcome $X = x_i$ is

$$\mathsf{i}(x_i) = \log_a \frac{1}{p_i}. \tag{2.5}$$

The base aÂ of the logarithm in (2.5) determines the unit of information. Typically, the base is 2, in which case the unit of information is the *bit* (contraction of *binary digit*), the fundamental unit of information. If we use the natural logarithm (base e), then the information unit is called *nat*. For notational simplicity, in the remainder of this chapter and in Chapter 3 we will use log (instead of $\log_2$) to denote the logarithm with base 2.

To demonstrate that this definition of information is appropriate, let's verify that it satisfies the three properties above. Obviously it satisfies P1: If two symbols x_i and x_j are equally probable, then the corresponding information contents $\mathsf{i}(x_i)$ and $\mathsf{i}(x_j)$ are also equal, i.e., the information content does not depend on the symbol itself, but only on its probability. P2 is also verified, since the logarithmic function is monotonically increasing with its argument. Finally, for P3 consider the following example.

Example 2.3 Consider an information source that outputs two independent symbols x and y. The information content of the outcome x, y is

$$\mathsf{i}(x, y) = \log \frac{1}{P(x, y)} = \log \frac{1}{P(x)P(y)} = \log \frac{1}{P(x)} + \log \frac{1}{P(y)} = \mathsf{i}(x) + \mathsf{i}(y),$$

so it satisfies P3.

Definition 2.1 defines the information content of each symbol of a source. A complete characterization of the source (equivalently, of the source alphabet $\mathcal{X}$ and the corresponding PMF $P_X(x)$) can then be obtained by defining the average information content, i.e., averaging over all source symbols,

$$\mathsf{H}(X) \triangleq \sum_{i=1}^{M} p_i \mathsf{i}(x_i) = \sum_{i=1}^{M} p_i \log \frac{1}{p_i}. \tag{2.6}$$

We can now formally define the measure of information of a source, which Shannon called **entropy**, due to its relationship to the concept of entropy in physics.

Definition 2.2 (Entropy) The entropy of a random variable (random symbol) X that takes values on the alphabet $\mathcal{X} = \{x_1, x_2, \dots, x_M\}$ with probabilities $p_1, p_2, \dots, p_M$ is defined as

$$\mathsf{H}(X) \triangleq \sum_{x \in \mathcal{X}} P(x) \log \frac{1}{P(x)} = \sum_{i=1}^{M} p_i \log \frac{1}{p_i}. \tag{2.7}$$

The entropy measures the *average information content* (or *uncertainty*) of X.

Remark 2.1 In (2.7) we need to consider the case where $p_i = 0$, which leads to an indeterminate. Luckily, we can use l'Hôpital's rule to solve it. Using $\log x = \frac{\ln x}{\ln 2}$, we can write

$$x \log \frac{1}{x} = \frac{1}{\ln 2} \cdot x \ln \frac{1}{x} = -\frac{1}{\ln 2} \cdot x \ln x = -\frac{1}{\ln 2} \cdot \frac{\ln x}{1/x},$$

where $\frac{\ln x}{1/x}$ is an indeterminate of the type $\frac{\infty}{\infty}$. We can apply now l'Hôpital's rule, $\lim_{x \to a} \frac{f(x)}{g(x)} = \lim_{x \to a} \frac{f'(x)}{g'(x)}$,

$$\lim_{x \to 0} \frac{\ln x}{1/x} = \lim_{x \to 0} \frac{1/x}{-1/x^2} = \lim_{x \to 0} (-x) = 0.$$

Thus, we don't need to worry about the case $p_i = 0$.

Example 2.4 Consider an information source that outputs symbols from the alphabet $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ with probabilities $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{4}$, $p_3 = p_4 = \frac{1}{8}$. The entropy of the source is

$$\mathsf{H}(X) = \frac{1}{2} \log 2 + \frac{1}{4} \log 4 + 2\frac{1}{8} \log 8 = 1.75 \text{ bits}.$$

2.2 The Binary Entropy Function

A special case of the entropy function is that of a binary source, i.e., $\mathcal{X} = \{x_1, x_2\}$. In this case, the entropy function is called the **binary entropy function**.

Definition 2.3 (Binary entropy function) Let X be a binary source with two possible outcomes $\mathcal{X} = \{x_1, x_2\}$ such that $P(x_1) = p$ and $P(x_2) = 1 - p$. Then

$$\mathsf{H}(X) = \mathsf{H}_\mathsf{b}(p),$$

where $\mathsf{H}_\mathsf{b}(p)$ is called the *binary entropy function* and is defined as

$$\mathsf{H}_\mathsf{b}(p) \triangleq p \log \frac{1}{p} + (1 - p) \log \frac{1}{1 - p}. \tag{2.8}$$

The function $\mathsf{H}_\mathsf{b}(p)$ is shown in Figure 2.1. It can be observed that it reaches a maximum for $p = 0.5$, i.e., the uncertainty is maximum when the two source symbols are equally likely. This result, i.e., the fact that the entropy attains its maximum for equiprobable symbols, is a general property of the entropy. The following lemma gives a lower and upper bound on the entropy of a random variable X.

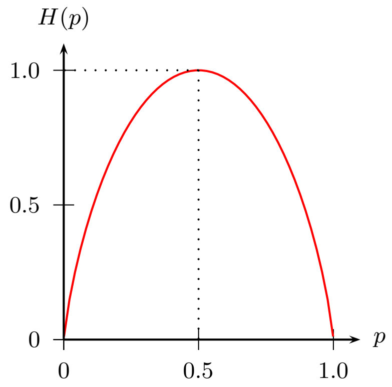


Figure 2.1: The binary entropy function as a function of the probability p.

Lemma 2.1 *The entropy of a random variable X that takes values on the alphabet $\mathcal{X} = \{x_1, \ldots, x_M\}$, $|\mathcal{X}| = M$, is bounded by*

$$0 \leq \mathsf{H}(X) \leq \log M \quad \textit{bits},$$

where

$$\begin{aligned} \mathsf{H}(X) &= 0 \qquad && \textit{if and only if} \;\; p_i = 1 \;\; \textit{for some} \;\; i, \\ \mathsf{H}(X) &= \log M \qquad && \textit{if and only if} \;\; p_i = \frac{1}{M} \;\; \forall i. \end{aligned}$$

Proof: We prove first the left inequality. We recall that $\mathsf{H}(X) = \sum_{i=1}^{M} p_i \log \frac{1}{p_i}$. Since $0 \leq p_i \leq 1$, we have that

$$p_i \log \frac{1}{p_i} \begin{cases} = 0 & \text{if } p_i = 0 \text{ or } p_i = 1 \\ > 0 & \text{if } 0 < p_i < 1 \end{cases}.$$

Thus, we have $\mathsf{H}(X) \geq 0$. Equality can only be achieved if and only if $p_i \log \frac{1}{p_i} = 0$ for all i, which implies $p_i = 1$ for exactly one i and $p_i = 0$ for the rest (in this case there is no uncertainty about the outcome of the source, therefore the information content is zero!).

We prove the right inequality by taking the difference $\mathsf{H}(X) - \log M$ and proving that it is less than or equal to zero. We will also use the following useful inequality,

$$\ln x \leq x - 1. \tag{2.9}$$

We can now proceed with the proof,

$$\begin{aligned}
\mathsf{H}(X) - \log M &= \sum_{i=1}^{M} p_i \log \frac{1}{p_i} - \log M \\
&= \sum_{i=1}^{M} p_i \log \frac{1}{p_i} - \log M \underbrace{\sum_{i=1}^{M} p_i}_{=1} \\
&= \sum_{i=1}^{M} p_i \log \frac{1}{p_i} - \sum_{i=1}^{M} p_i \log M \\
&= \sum_{i=1}^{M} p_i \log \frac{1}{M p_i} \\
&\overset{(a)}{=} \frac{1}{\ln 2} \sum_{i=1}^{M} p_i \ln \frac{1}{M p_i} \\
&\overset{(b)}{\leq} \frac{1}{\ln 2} \sum_{i=1}^{M} p_i \left(\frac{1}{M p_i} - 1 \right) \\
&= \frac{1}{\ln 2} \left(\sum_{i=1}^{M} \frac{1}{M} - \sum_{i=1}^{M} p_i \right) \\
&= \frac{1}{\ln 2} (1 - 1) = 0,
\end{aligned}$$

where in (a) we used the fact that $\log(x) = \frac{\ln x}{\ln 2}$ and in (b) we used (2.9). Thus, $\mathsf{H}(X) \leq \log M$, with equality if and only if $p_i = \frac{1}{M}$ for all i. ■

Chapter 3

Data Compression

In the previous chapter, we have introduced a natural measure of the (average) information content of a discrete source. We have seen that the output of an information source is a sequence of random variables $X^{(1)}X^{(2)}X^{(3)}\ldots$. Therefore, the information content of the source corresponds to the information content of the random variable X, the entropy $\mathsf{H}(X)$. We have also seen that improbable outcomes convey more information than probable ones. The entropy can therefore be seen as a measure of uncertainty.

An important problem in communications is the **efficient representation of the data generated by a source**. A common characteristic of information sources is that they contain redundancy, i.e., information that is unnecessary and whose transmission is a waste of resources. Think for example about the written English, where individual letters can be mistyped or dropped and the text usually remains quite meaningful. As an example, consider the sentence "C_n y_u reco_er this q_est_on?". Despite missing some letters the text is still meaningful because it contains a lot of redundant information! Thus, we would like to find an efficient representation of the output of the source (individual symbols or sequences of symbols, called messages) which results **in zero or little redundancy**, i.e., using as few digits as possible. In this chapter, we discuss this fundamental problem.

The process by which an efficient representation of the source is accomplished is called **source compression**. Source compression is achieved by encoding (mapping) the symbols (or messages) at the output of the source to a sequence of digits, called **codeword**, that is **shorter** in average. Therefore, source compression is also typically referred to as **source coding**, and the device that maps the source output into codewords is known as **source encoder**. In this course, with no loss of generality, we will restrict ourselves to source compression using binary codes, i.e., the output of the source is encoded into a sequence of bits.

We start discussing source encoders that **encode one source symbol at a time**. For the source encoder to be efficient, we typically require knowledge of the statistics of the source. In particular, if the symbols generated by a source are not equiprobable, we can take advantage of this feature to compress the source. Intuitively, we can achieve compression *on average* by assigning shorter codewords to more probable symbols and longer codewords to less likely ones. We refer to a source encoder that assigns codewords of different lengths to source symbols, as a **variable-length encoder**.

Several questions come to us: **How much can we compress such that we lose no information? Are there efficient methods to assign codewords to source symbols? How can we make sure that the source code is easy to decode?**

Let's start to clarify these concepts with a motivating example. Consider a primitive alphabet with only

Table 3.1: Different source codes to compress an alphabet with four letters.

symbol	probability	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$
a	1/2	000	00	00	0	0	0
b	1/4	001	00	01	01	01	10
c	1/8	101	10	10	001	011	110
d	1/8	111	11	11	111	111	111

four letters, a, b, c, d, with probabilities $\Pr(a) = 1/2$, $\Pr(b) = 1/4$, $\Pr(c) = 1/8$, and $\Pr(d) = 1/8$. We would like to design a good (binary) source code to compress the language produced by this alphabet. Consider the codes $\mathcal{C}_i$, $i = 1, \ldots, 6$, in Table 1.

1. $\mathcal{C}_1$ is a valid code. However, we can definitely represent the source more efficiently, since considering fixed-length coding (i.e., we encode each letter to a codeword of the same length), the minimum number of bits we require to represent 4 symbols is $\log 4 = 2$.

2. $\mathcal{C}_2$ assigns shorter codewords to each symbol. However, it is useless, because the same codeword 00 is used for two symbols: 00 can be decoded into a or b. A code that assigns the same codeword to more than one symbol is called *singular*. Thus, for a code to be useful, it must be *nonsingular*.

3. $\mathcal{C}_3$ has the same average length of $\mathcal{C}_2$ and is a useful code. The question is: Can we find a more *efficient* code, i.e., a code with average length smaller than 2? The answer is yes: We can do better by considering the source statistics and constructing a variable-length code. Since a is the most likely symbol, it seems reasonable to be encoded to the shortest codeword.

4. Consider the code $\mathcal{C}_4$. At a first glance it is a better code, since the average codeword length is smaller (see Example 3.3 later on). However, let's look at it more closely. Assume that we receive the binary sequence 001. It can be decoded to either ab or c. Therefore, this is not a good code; the code is not uniquely decodable, and is useless in practice.

5. $\mathcal{C}_5$ is an optimal code, in the sense that it is uniquely decodable and achieves the smallest average codeword length. However, it is not so "easy" to decode. For instance, if we receive the string 01 we cannot decide which symbol was transmitted, until more bits are received. If we receive an extra bit, say 0, i.e., we have received 010, we can conclude that the first two bits correspond to the letter b, but we cannot arrive to this conclusion before.

6. $\mathcal{C}_6$ is a better code: It is uniquely decodable, achieves the smallest average codeword length, and is easier to decode.

Example 3.1 A famous example of variable-length coding is the Morse code. The Morse code assigns (encodes) the most frequent letters to shorter codewords, and less frequent letters to longer codewords. In English, the letter "e" is the most frequent one, while "q" is the least frequent one. For this reason, the Morse code encodes the letter "e" to a single dot "$\cdot$", i.e., it assigns to the letter "e" the shortest codeword in the code, and "q" to the codeword "$- - \cdot -$", the longest one in the code.

We can now establish some properties that a source code should satisfy.

P1 The code must be **uniquely decodable**, i.e., the symbols generated by the source must be uniquely retrieved from the encoded string of bits.

P2 The code should be easy to decode.

P3 The code should compress the source as much as possible.

In the following we address these three properties.

The following definition will be useful.

Definition 3.1 Given a set Σ, Σ^+ denotes the set of all strings over Σ of any (nonzero) finite length.

For example, if $\Sigma = \{a, b, c\}$ then a, ab, aac and $bbac$ are strings over Σ of lengths one, two, three and four respectively.

We can now formally define a (binary) source encoder.

Definition 3.2 (Binary source encoder and code) Consider a random variable X which takes values on $\mathcal{X} = \{x_1, \ldots, x_M\}$. A binary encoder $\mathcal{E}$ is a function $\mathcal{E} : \mathcal{X} \to \{0,1\}^+$ that maps each source symbol $x_i \in \mathcal{X}$ to a binary sequence $\mathbf{c}_i \in \{0,1\}^+$, called *codeword*. The number of bits in $\mathbf{c}_i$ is called its *length* and is denoted by ℓ_i. The binary source code $\mathcal{C}$ is the set of all codewords, i.e., $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_M\}$.

Remark 3.1 We will sometimes write $\mathbf{c}(x)$ to emphasize that $\mathbf{c}$ is the codeword associated to symbol x.

We assume that successive codewords generated by a variable-length encoder $\mathcal{E}$ are assumed to be transmitted as a continuing sequence of bits (without any marks that determine the end and the beginning of a codeword).

We also define the *extended* code as follows.

Definition 3.3 (Extended code) The extended code $\mathcal{C}^+$ is the set of binary sequences resulting from the mapping $\mathcal{E}^+ : \mathcal{X}^+ \to \{0,1\}^+$ from a sequence of symbols that take values on the alphabet $\mathcal{X}$ to a binary sequence obtained by concatenating the corresponding codewords. For a sequence of n source symbols $x^{(1)}, \ldots, x^{(n)}$, at times $t = 1, \ldots, n$, the extended code sequence is

$$\mathbf{c}^+(x^{(1)}x^{(2)} \ldots x^{(n)}) = \mathbf{c}(x^{(1)})\mathbf{c}(x^{(2)}) \ldots \mathbf{c}(x^{(n)}),$$

where, as before, the superindex indicates time.

3.1 Uniquely Decodable Codes and Prefix-Free Codes

At the receiver side, given a sequence of bits, the source decoder must determine which symbols were transmitted, which amounts to identifying the transmitted codewords within the sequence of bits (i.e., we need to determine when the codewords start and end) and then undo the mapping $\mathcal{E}$. Obviously, we would like to uniquely identify the sequence of symbols generated by the source from the received sequence of bits. We are now ready to formally define property P1 (**unique decodability**).

Definition 3.4 (Uniquely decodable code) A code $\mathcal{C}$ is uniquely decodable if, for any two different sequences of symbols $\boldsymbol{x} = x^{(1)}x^{(2)}\dots x^{(n)}$ and $\boldsymbol{y} = y^{(1)}y^{(2)}\dots y^{(n)}$, the corresponding concatenation of codewords $\mathbf{c}^{+}(\boldsymbol{x}) = \mathbf{c}(x^{(1)})\mathbf{c}(x^{(2)})\dots\mathbf{c}(x^{(n)})$ and $\mathbf{c}^{+}(\boldsymbol{y}) = \mathbf{c}(y^{(1)})\mathbf{c}(y^{(2)})\dots\mathbf{c}(y^{(n)})$ are different, i.e.,

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}^{+}, \boldsymbol{x} \neq \boldsymbol{y} \Longrightarrow \mathbf{c}^{+}(\boldsymbol{x}) \neq \mathbf{c}^{+}(\boldsymbol{y}).$$

In plain words, any sequence of source symbols can be unambiguously recovered from the corresponding code sequence $\mathbf{c}^{+}$.

Now let's consider property P2. In general, decoding a uniquely decodable code may be quite complicated. However, it is desirable that the code is "easy" to decode. Here we say that a code is easy to decode if we can decode each codeword immediately at the arrival of the last bit of that codeword, i.e., we need to be able to identify the end of a codeword as soon as the last bit of the codeword is received. In practice, this means that **no codeword can be a prefix of another codeword**.

Example 3.2 Assume transmission using the codes $\mathcal{C}_5$ and $\mathcal{C}_6$ in Table 3.1, which are both uniquely decodable. Using $\mathcal{C}_6$, as soon as we receive 0 we identify that the transmitted symbol was a. However, for code $\mathcal{C}_5$, if we receive 0 we do not know yet which symbol was transmitted. We need to wait until more bits are received: If the received sequence is 001, then the first zero would be decoded to a. However, if the received sequence is 011, it would be decoded to c. Therefore, for $\mathcal{C}_5$ decoding is a bit *harder*: we need to wait for some extra bits in order to decode.

This leads us to the important concept of **prefix-free codes**.

Definition 3.5 (Prefix-free code) A code is called a prefix-free code if no codeword is the prefix of any other codeword.

Note that for a prefix-free code we can decode codewords instantaneously, i.e., as soon as they are received (the end of a codeword is recognized immediately), without the need to wait for additional received bits. For this reason, a prefix-free code is sometimes also called an **instantaneous code**. It is important to remark that **a prefix-free code is always uniquely decodable**.

3.1.1 Binary Code Tree

A very useful graphical representation of a prefix-free code is that of a binary tree where codewords correspond to leaves. In general, any binary code can be represented by a binary tree which grows from the root to its leaves, such that each node in the tree has two descendants, i.e., it divides into two *branches*, where each branch is labeled with 0 or 1. Each node in the tree represents the binary string corresponding to the branches from the root to the node. Then, every codeword can be represented by a particular path through the tree. In the particular case of prefix-tree codes, every codeword corresponds to a leaf, in the sense that every codeword can be read as the binary sequence labeling the branches of the tree from the root to a leaf, while intermediate nodes (i.e., nodes dividing into branches) correspond to prefixes of some codewords.

The binary tree of the prefix-free code $\mathcal{C}_6$ in Table 3.1 is given in Figure 3.1. Starting from the root, the two branches that lead to the nodes at depth 1, labeled by 0 and 1, correspond to the first bit of the codeword. The two branches departing from the nodes at depth 1 correspond to the second bit of the codeword, and so on. The codeword associated to the symbol d corresponds to one of the leaves in the tree: The codeword associated to symbol d is the binary sequence obtained by traversing the tree from its root to the bottom leaf of the tree, i.e., the sequence 111.

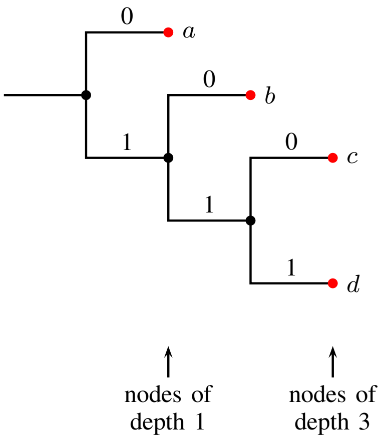


Figure 3.1: Binary tree of the prefix-free code $\mathcal{C}_6$ in Table 3.1. The red nodes indicate leaves of the tree, and correspond to codewords.

Note that since codewords correspond exclusively to leaves in the tree, no codeword is a prefix of another codeword.

3.1.2 Kraft's Inequality

We have seen that prefix-free codes are uniquely decodable. The following theorem gives a sufficient and necessary condition on the existence of a prefix-free code.

Theorem 3.1 (Kraft's inequality) *There exists a binary prefix-free code of cardinality M and codeword lengths $\ell_1, \ell_2, \dots, \ell_M$ if and only if*

$$\sum_{i=1}^{M} 2^{-\ell_i} \leq 1. \tag{3.1}$$

Proof: We first prove necessity ($\Longrightarrow$). Assume that there exists a prefix-free code. Since the code is prefix-free, it can be described by a binary tree, where each codeword, of length ℓ_i, corresponds to a leaf of the tree at depth ℓ_i. Let $\ell_{\mathsf{max}} = \max(\ell_1, \ell_2, \dots, \ell_M)$, and expand the tree so that all branches have depth ℓ_{max}. If expanded, a leaf (codeword) at depth ℓ_i would lead to $2^{\ell_{\mathsf{max}}-\ell_i}$ leaves at depth ℓ_{max}. The total number of leaves at depth ℓ_{max} after the expansion of all codewords must be less than or equal to $2^{\ell_{\mathsf{max}}}$. Thus,

$$\sum_{i=1}^{M} 2^{\ell_{\mathsf{max}}-\ell_i} \leq 2^{\ell_{\mathsf{max}}}.$$

Dividing both sides by $2^{\ell_{\mathsf{max}}}$ we finally obtain

$$\sum_{i=1}^{M} 2^{-\ell_i} \leq 1.$$

We now prove sufficiency ($\Longleftarrow$), i.e., assuming $\sum_{i=1}^{M} 2^{-\ell_i} \leq 1$, we need to prove that we can always construct a prefix-free code or, in other words, a binary tree code where codewords correspond to leaves. Assume without loss of generality that the codeword lengths are sorted in increasing order, $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_M$. We can construct a prefix-free code as follows.

Step1 Start with a complete tree where all leaves are at depth ℓ_M. For each node at depth i we have two nodes at depth $i+1$.

Step2 Choose a free node at depth ℓ_1 for the first codeword, and remove all its descendants (this removes $2^{\ell_M-\ell_1}$ leaves at depth ℓ_M). Choose a free node at depth ℓ_2 for codeword 2. This removes $2^{\ell_M-\ell_2}$ leaves at depth ℓ_M. Continue for depths $\ell_3, \ell_4, \ldots$ until we have placed all codewords.

In order for the algorithm to work, we require that at every step i there are free leaves to be assigned to a codeword. After assigning a node at depth ℓ_j to a codeword, the number of removed leaves at depth ℓ_M is $\sum_{i=1}^{j} 2^{\ell_M-\ell_i}$, or, in other words, the number of remaining leaves at depth M is

$$2^{\ell_M} - \sum_{i=1}^{j} 2^{\ell_M-\ell_i}.$$

Now we can use the fact that $\sum_{i=1}^{j} 2^{-\ell_i} < \sum_{i=1}^{M} 2^{-\ell_i}$ for $j < M$, and write

$$2^{\ell_M} - \sum_{i=1}^{j} 2^{\ell_M-\ell_i} = 2^{\ell_M}\left(1 - \sum_{i=1}^{j} 2^{-\ell_i}\right) > 2^{\ell_M}\left(1 - \sum_{i=1}^{M} 2^{-\ell_i}\right) \geq 0,$$

where for the last inequality we used the fact that Kraft's inequality is fulfilled.

This shows that there are free leaves in every step. Thus, we can construct a prefix-free code with the given codeword lengths. ■

Up to now we have focused on prefix-free codes, because they are uniquely decodable and easy to decode. The question is: Is there a uniquely-decodable code that is not prefix-free that achieves better performance (i.e., lower average codeword length)? In other words, do we lose any optimality by restricting ourselves to prefix-free codes? The answer to this question is given in the following theorem.

Theorem 3.2 (Kraft-McMillan's inequality) *A uniquely decodable code with codeword lengths $\ell_1, \ldots, \ell_M$ exists if and only if*

$$\sum_{i=1}^{M} 2^{-\ell_i} \leq 1. \tag{3.2}$$

We know from Theorem 3.1 that every prefix-free code also satisfies (3.2). Therefore, for any non-prefix-free code with codeword lengths that satisfy (3.2) we can find a prefix-free code with the same codeword lengths, hence giving the same performance. The result of Theorem 3.2 is very important: It tells us that we can restrict ourselves to prefix-free codes (which are easy to decode) without losing in performance.

3.2 Efficient Codes

We address now property P3. Obviously, we would like the code to be able to efficiently represent the source symbols, i.e., we would like to compress the source as much as possible. Our goal will therefore be to minimize the average number of code bits per source symbol, i.e., the average codeword length,

$$\bar{L} = \mathbb{E}[L] = \sum_{i=1}^{M} p_i \ell_i. \tag{3.3}$$

where p_i is the probability that the source emits symbol x_i, and ℓ_i is the length of the corresponding codeword $\mathbf{c}_i$ (see Definition 3.2).

Example 3.3 Code $\mathcal{C}_5$ in Table 3.1 has average codeword length

$$\bar{L} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + 2 \cdot \frac{1}{8} \cdot 3 = 1.75,$$

i.e., it is more efficient than using a fixed-length code that assigns length-2 codewords to symbols.

3.3 The Limits of Source Coding

We can now reconsider the symbol probabilities $\{p_i\}$. For a given PMF P_X, how can we assign codeword lengths to achieve the best compression (i.e., to minimize $\bar{L}$)? and what's the best achievable compression? We address these questions in this and next section.

The goal is to minimize the average codeword length

$$\bar{L} = \sum_{i=1}^{M} p_i \ell_i.$$

As you may have guessed already, the entropy $\mathsf{H}(X)$ gives a lower bound on the average codeword length.

Theorem 3.3 *The average codeword length of a uniquely decodable code is lower bounded by*

$$\bar{L} \geq \mathsf{H}(X).$$

Proof: Consider the difference between the entropy and the average codeword length,

$$\begin{aligned}
\mathsf{H}(X) - \bar{L} &= \sum_{i=1}^{M} p_i \log \frac{1}{p_i} - \sum_{i=1}^{M} p_i \ell_i \\
&= \sum_{i=1}^{M} p_i \left[\log \frac{1}{p_i} - \log 2^{\ell_i} \right] = \sum_{i=1}^{M} p_i \left[\log \frac{1}{p_i} + \log 2^{-\ell_i} \right] \\
&= \sum_{i=1}^{M} p_i \log \frac{2^{-\ell_i}}{p_i} \\
&= \frac{1}{\ln 2} \sum_{i=1}^{M} p_i \ln \frac{2^{-\ell_i}}{p_i} \\
&\overset{(a)}{\leq} \frac{1}{\ln 2} \sum_{i=1}^{M} p_i \left(\frac{2^{-\ell_i}}{p_i} - 1 \right) \\
&= \frac{1}{\ln 2} \left(\sum_{i=1}^{M} 2^{-\ell_i} - \underbrace{\sum_{i=1}^{M} p_i}_{=1} \right) = \frac{1}{\ln 2} \left(\sum_{i=1}^{M} 2^{-\ell_i} - 1 \right) \overset{(b)}{\leq} 0,
\end{aligned}$$

where in (a) we used (2.9) and in (b) we used Kraft's inequality (see (3.1)). ■

Comparing the definition of the entropy (see (2.7) in Definition 2.2) and (3.3), we observe that $\bar{L} = \mathsf{H}(X)$ if and only if the codeword lengths satisfy $\ell_i = \log \frac{1}{p_i}$ $\forall i$.

Theorem 3.3 shows that we cannot compress the source below its entropy, i.e., below its information content. In general, $\ell_i = \log \frac{1}{p_i}$ is not satisfied for all i. Therefore, we cannot expect to achieve the lower bound $\mathsf{H}(X)$. However, how close to the entropy can we expect to compress? The following theorem gives an answer to this question.

Theorem 3.4 (Source Coding Theorem for a single random symbol) *Let X be a random variable (random symbol) generated by a discrete memoryless source with entropy $\mathsf{H}(X)$. There exists a prefix-free code $\mathcal{C}$ with average codeword length satisfying*

$$\mathsf{H}(X) \leq \bar{L}(\mathcal{C}) < \mathsf{H}(X) + 1. \tag{3.4}$$

Proof: We set the codeword lengths to integers slightly larger than the optimum lengths,

$$\ell_i = \left\lceil \log \frac{1}{p_i} \right\rceil, \tag{3.5}$$

where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. We check that there is a prefix-free code with these lengths by confirming that the Kraft inequality is satisfied,

$$\sum_{i=1}^{M} 2^{-\ell_i} = \sum_{i=1}^{M} 2^{-\left\lceil \log \frac{1}{p_i} \right\rceil} \leq \sum_{i=1}^{M} 2^{-\log \frac{1}{p_i}} = \sum_{i=1}^{M} p_i = 1.$$

Therefore,

$$\bar{L}(\mathcal{C}) = \sum_{i=1}^{M} p_i \left\lceil \log \frac{1}{p_i} \right\rceil < \sum_{i=1}^{M} p_i \left(\log \frac{1}{p_i} + 1 \right) = \sum_{i=1}^{M} p_i \log \frac{1}{p_i} + \sum_{i=1}^{M} p_i = \mathsf{H}(X) + 1.$$

■

3.4 Optimal Source Coding: Huffman Coding

Given a PMF P_X, **how can we design an optimal prefix-free code**? Here, by optimal code we mean that, constraining the code to be prefix-free, the average codeword length is minimized (i.e., no other code with a lower average codeword length exists). We now present a simple algorithm for finding an optimal prefix-free code, the **Huffman Coding Algorithm**, proposed by David A. Huffman during his PhD at MIT as a solution to a term paper. Huffman's main idea was to construct the code *backwards*, starting from the tails of the codewords (equivalently, building the tree from the leaves to the root).

The Huffman Coding Algorithm for Optimal Binary Codes

1. Order the M source symbols in decreasing order of their probabilities. Create M leaves corresponding to the M possible symbols and assign their probabilities to them.

2. Combine the two least probable symbols into a single one, with probability equal to the sum of probabilities. This amounts to taking the two leaves corresponding to the two least probable symbols of the alphabet, create a new node that has these two leaves as descendants, and assigning the sum of probabilities to the node.

3. Repeat Step 2 until only one node is free, and root it.

4. Starting from the root of the obtained tree, assign the binary symbols 0 and 1 to each pair of branches that arise from each node. The codeword for each symbol is read as the binary sequence from the root to the leaf associated to the symbol.

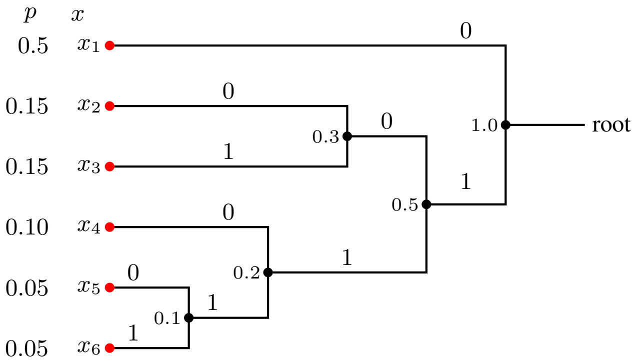


Figure 3.2: Binary tree resulting from applying the Huffman algorithm to the source with six symbols of Example 3.4.

Table 3.2: Huffman code of Figure 3.2

symbol	codeword
x_1	0
x_2	100
x_3	101
x_4	110
x_5	1110
x_6	1111

Example 3.4 Consider a source alphabet consisting of 6 symbols $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, with probabilities $\{0.5, 0.15, 0.15, 0.10, 0.05, 0.05\}$. The entropy is $\mathsf{H}(X) = 2.08548$ bits. An optimal prefix-free code can be obtained by applying Huffman's algorithm as shown in Figure 3.2. The code is given in Table 3.2. The average codeword length is $\bar{L} = 2.1$ bits, and no prefix-free code with smaller $\bar{L}$ can be constructed.

3.5 Encoding Blocks of Symbols

In the previous sections we have considered source compression by **encoding each output of the source independently**. The question is whether there is a more efficient approach to compress a source. In this section, we show that **encoding blocks of symbols we can indeed do better than by encoding each symbol separately.**

Consider combining ν consecutive symbols at the output of a source X, $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$, in a new symbol (which we will refer to as **message**),

$$(x^{(1)}, x^{(2)}, \ldots, x^{(\nu)}),$$

and encode this message using a Huffman code. In other words, we design a code for an equivalent source Y with a new finite alphabet $\mathcal{Y} \equiv \{\mathcal{X}\}^\nu$ which contains M^ν messages, $y_1, y_2, \ldots, y_{M^\nu}$, consisting of all possible tuples $(x^{(1)}, x^{(2)}, \ldots, x^{(\nu)})$, $x^{(j)} \in \mathcal{X}$, of length ν. Since the source is memoryless, the probability of a message

y_i is

$$\boxed{p_i = \prod_{j=1}^{\nu} p_{i,j},}$$

where $p_{i,j}$ is the probability of the j-th symbol of y_i. We know from Theorem 3.4 that we can build a code for Y with average length $\bar{L}_\nu$,

$$\boxed{\mathsf{H}(Y) \leq \bar{L}_\nu < \mathsf{H}(Y) + 1,} \tag{3.6}$$

where we used the subindex ν to stress that we encode messages containing ν symbols.

Note that the average codeword length $\bar{L}$ for the code that encodes source symbols independently corresponds to the case $\nu = 1$, in which case $Y = X$. Furthermore, note that we cannot directly compare $\bar{L}_\nu$ for different values of ν, because for larger ν, $\bar{L}_\nu$ will obviously be larger. A fair comparison is to compare the average codeword length necessary to describe one source symbol. Since a message $y \in \mathcal{Y}$ contains ν symbols $x^{(1)}, \ldots, x^{(\nu)} \in \mathcal{X}$, $\bar{L}_\nu/\nu$ gives the right measure to compare codes. We can write

$$\frac{\mathsf{H}(Y)}{\nu} \leq \frac{\bar{L}_\nu}{\nu} < \frac{\mathsf{H}(Y) + 1}{\nu}. \tag{3.7}$$

Now, since the random message Y consists of independent and identically distributed random variables X (i.e., each message in $\mathcal{Y}$ is built from ν independent symbols of the original alphabet $\mathcal{X}$),[1]

$$\mathsf{H}(Y) = \nu \mathsf{H}(X).$$

Thus,

$$\mathsf{H}(X) \leq \frac{\bar{L}_\nu}{\nu} < \mathsf{H}(X) + \frac{1}{\nu}. \tag{3.8}$$

We can now formulate Shannon's Source Coding Theorem.

Theorem 3.5 (Source Coding Theorem) *Let X be a random variable (random symbol) generated by a discrete memoryless source with entropy $\mathsf{H}(X)$. There exists a prefix-free code $\mathcal{C}$ that results from the encoding of messages of length ν symbols with average codeword length per source symbol $\frac{\bar{L}_\nu}{\nu}$ satisfying*

$$\mathsf{H}(X) \leq \frac{\bar{L}_\nu}{\nu} < \mathsf{H}(X) + \frac{1}{\nu}. \tag{3.9}$$

A very important consequence from Theorem 3.5 is that choosing ν large enough we can approach the ultimate limit of compression, $\mathsf{H}(X)$, arbitrarily closely using Huffman codes! However, this comes at a cost: Approaching $\mathsf{H}(X)$ requires encoding very large blocks of symbols, i.e., it requires long codes, which are difficult to construct. Furthermore, this may introduces a large delay since the source encoder can only encode after receiving a large number of source symbols.

We can use the Source Coding Theorem to derive a measure of the efficiency of a source code. Since from Theorem 3.5 it follows that $\nu \mathsf{H}(X) \leq \bar{L}_\nu$, we can write $\frac{\nu \mathsf{H}(X)}{\bar{L}_\nu} \leq 1$ and define the efficiency of a source code as

$$\boxed{\eta = \frac{\nu \mathsf{H}(X)}{\bar{L}_\nu},} \tag{3.10}$$

where $0 \leq \eta \leq 1$.

[1] Showing this equality is left as an exercise.

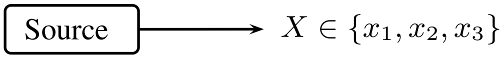


Figure 3.3: Source X of Example 3.5.

Table 3.3: Huffman code of Example 3.5, encoding symbols separately, $\bar{L} = 1.55$, $\eta = 0.976$.

symbol	probability	information content	codeword
x_1	0.45	1.152	1
x_2	0.35	1.514	00
x_3	0.20	2.321	01

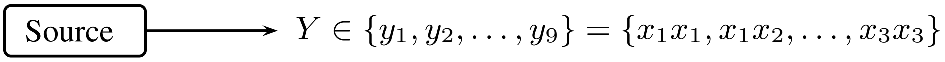

Source
$Y \in \{y_1, y_2, \dots, y_9\} = \{x_1x_1, x_1x_2, \dots, x_3x_3\}$

Figure 3.4: Equivalent source Y of Example 3.5, encoding blocks of two symbols.

Example 3.5 Consider the discrete memoryless source in Figure 3.3, whose output X takes values on the alphabet $\mathcal{X} = \{x_1, x_2, x_3\}$, with probabilities $\{0.45, 0.35, 0.20\}$. The entropy of the source is $\mathsf{H}(X) = 1.513$ bits. The Huffman code for this source is given in Table 3.3. It has an average codeword length $\bar{L} = 1.55$ bits, which results in an efficiency $\eta = 0.976$.

Alternatively we can encode groups of symbols. Consider the case where we encode pairs of symbols. We thus define the random message $Y = (X^1, X^2)$. We also define the new alphabet $\mathcal{Y} = \{y_1, y_2, \dots, y_9\} = \{x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3\}$, of cardinality $|\mathcal{Y}| = |\mathcal{X}|^2 = 9$. The equivalent source is depicted in Figure 3.4. The probabilities of the symbols $y_1, y_2, \dots, y_9$ are given in Table 3.4. The entropy of the random message Y is $\mathsf{H}(Y) = 2\mathsf{H}(X) = 3.026$ bits. The Huffman code, given also in Table 3.4, has $\bar{L}_\nu = 3.0675$. Normalizing this value by ν we can compare the compression achieved by applying Huffman to the source Y with that of the Huffman code encoding symbols separately: $\frac{\bar{L}_\nu}{\nu} = 1.53375$ as compared to 1.55, i.e., we achieve a higher compression. The efficiency of the code is now $\eta = 0.986$.

3.6 Universal Data Compression: The Lempel-Ziv Algorithm

The source compression discussed above suffers from a main drawback: It requires knowledge of the source statistics. In practice, the statistics of the source may not be available (e.g., what is the statistic of different kinds of music?). Furthermore, ideally we would like a compression algorithm to be independent of the source. A source compression algorithm that achieves the ultimate limit regardless of the source is called a **universal compression** algorithm. One of the most popular universal source compression algorithms is the Lempel-Ziv algorithm, introduced by Abraham Lempel and Jacob Ziv, which compresses the source without making any assumption on its probabilistic model. The Lempel-Ziv algorithm and its variants are widely used, e.g., in the gzip and gif compression standards, and can be proved to asymptotically compress down to the entropy of the source.

There are several variants of the Lempel-Ziv algorithm, but all follow the same basic idea. In the following, we discuss the LZ78 algorithm, as it is one of the simplest to explain (and analyze). The main idea is to

Table 3.4: Huffman code of Example 3.5, encoding blocks of two symbols, $\bar{L}_\nu = 3.0675$, $\eta = 0.989$.

symbol	probability	information content	codeword
x_1x_1	0.2025	2.304	10
x_1x_2	0.1575	2.666	001
x_2x_1	0.1575	2.666	010
x_2x_2	0.1225	3.039	011
x_1x_3	0.09	3.473	111
x_3x_1	0.09	3.473	0000
x_2x_3	0.07	3.836	0001
x_3x_2	0.07	3.836	1100
x_3x_3	0.04	4.643	1101

construct a dictionary of substrings, which are usually referred to as *phrases*, that appear in the text.

We will explain the LZ78 algorithm through a running example. Assume a binary source $\mathcal{X} \in \{A, B\}$ and the sequence

$$\text{ABBABBABBBAABABAABAAA} \tag{3.11}$$

To compress the source sequence, the algorithm works as follows.

1. Parse the source sequence into the shortest phrases that have not appeared before, i.e.,

   $$\text{A} \mid \text{B} \mid \text{BA} \mid \text{BB} \mid \text{AB} \mid \text{BBA} \mid \text{ABA} \mid \text{BAA} \mid \text{BAAA}$$

2. Enumerate the phrases sequentially starting from 1,

   $$\underbrace{\text{A}}_{1} \mid \underbrace{\text{B}}_{2} \mid \underbrace{\text{BA}}_{3} \mid \underbrace{\text{BB}}_{4} \mid \underbrace{\text{AB}}_{5} \mid \underbrace{\text{BBA}}_{6} \mid \underbrace{\text{ABA}}_{7} \mid \underbrace{\text{BAA}}_{8} \mid \underbrace{\text{BAAA}}_{9}$$

   We denote by n_p the number of distinct phrases. For this example, $n_\mathsf{p} = 9$.

3. Encode each phrase by the pair consisting of the number of its prefix phrase and the additional new symbol, as shown in the third column of Table 3.5.

4. Finally, encode the pairs (number, symbol) into binary codewords of length $\lceil \log_2(n_\mathsf{p}) \rceil + \lceil \log_2 |\mathcal{X}| \rceil$ by mapping the number to a sequence of $\lceil \log_2(n_\mathsf{p}) \rceil$ bits and the symbol to a sequence of $\lceil \log_2 |\mathcal{X}| \rceil$ bits. This is shown in the fourth column of Table 3.5. The binary sequence at the output of the encoder is thus,

   $$00000\;00001\;00100\;00101\;00011\;01000\;01010\;00110\;10000$$

Note that this sequence is longer than the original binary sequence in (3.11), i.e., the compression algorithm has not compressed at all but has expanded the original sequence! This is due to the fact that the sequence in this toy example is too short for the efficiency of the Lempel-Ziv algorithm to become noticeable.

Table 3.5

index	phrase	encoding	codeword
0	∅		
1	A	(∅,A)	0000 0
2	B	(∅,B)	0000 1
3	BA	(2,A)	0010 0
4	BB	(2,B)	0010 1
5	AB	(1,B)	0001 1
6	BBA	(4,A)	0100 0
7	ABA	(5,A)	0101 0
8	BAA	(3,A)	0011 0
9	BAAA	(8,A)	1000 0

3.7 Further Discussion and Remarks

Perhaps you have already realized that the source compression analyzed in this chapter is **lossless**, i.e., we can always recover the transmitted sequence of data symbols from the coded sequence with no loss of information. It is not surprising that in this case we cannot compress below the entropy of the source! Other compression methods, instead, compress below $\mathsf{H}(X)$ to achieve a larger compression. In this case we talk about **lossy source compression**. Obviously, by compressing below the entropy of the source we lose some information, but perhaps the loss is so little, or the information thrown away is so unimportant, that in practice there is no impact on performance. This is the case for example of algorithms to compress sound, video or image, e.g., the jpeg or the mp3 compression standards. Developing lossy compression algorithms is typically closely related to the human perception. For instance, there are some audio tones that are not audible to a human, or a picture may contain more detail than the human eye can distinguish. Therefore, we can remove this detail (i.e., compress the picture further) with no impact in the human's perception of the picture. Note that in this case, **after decompression we cannot recover the original data identically.**

Chapter 4

Communication over a Noisy Channel

The basic communication problem is the transmission of data over a noisy channel. In Chapters 2 and 3 we discussed how we can represent a discrete memoryless source efficiently. We have also implicitly assumed that the channel from the source compressor to the receiver was noise-free. In practice, however, the channel is noisy and is affected by distortion and other channel impairments, i.e., it introduces errors. In this chapter we wish to answer the following fundamental questions: **How much information can we transmit reliably over a noisy channel?** and **how do we achieve this in practice**? In other words, we would like to make a noisy channel behave as if it was noise-free. In 1948, Claude E. Shannon established that for many common classes of channels there exists **a fundamental limit, i.e., a highest rate, at which information can be transmitted reliably over the channel**. This fundamental limit is called the **channel capacity**.

To understand the fundamental limits of communication over a noisy channel we first need to define what a channel is. A channel is a physical medium through which the message (in our case a stream of bits) is sent from a sender (the source) to a receiver (the destination). Examples of channels are a satellite link, an optical fiber, the wireless channel, or a hard-disk drive (including the process of writing and reading to/from the disk). A channel is said to be noisy if the data at the channel output is not exactly the same as the data at the channel input (for example due to perturbations introduced by atmosphere turbulences in the case of a satellite link).

We will implicitly assume that the data to be transmitted has no redundancy, i.e., it has gone through a source compressor which has removed the inherent redundancy. Then, we transmit this information over a physical channel and we would like to recover it at the receiver side. If the channel is noiseless, the maximum information that we can convey over the channel is obviously the source entropy $\mathsf{H}(X)$. However, if the channel is noisy, it is not trivial to determine how much information we can convey through the channel. To address this problem, we need to introduce a mathematical model of a noisy communication channel.

4.1 Model of a Noisy Communication Channel

Picture a water pipe of a certain diameter. The capacity of the pipe is the maximum water flow that it can carry. Above this maximum flow, the pipe will collapse. A communication channel is similar to a water pipe. In this case, instead of carrying water, the channel serves to carry (transmit) information, i.e., (binary) symbols. A binary-input binary-output communication channel, and its analogy to a water pipe, is depicted

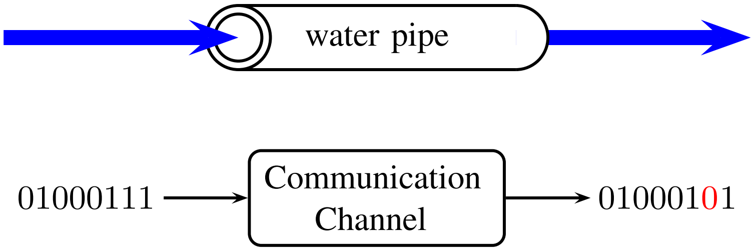


Figure 4.1: A binary-input binary-output communication channel and its analogy with a water pipe. The bit in red indicates that the channel has introduced one error.

in Figure 4.1.

Let $\boldsymbol{x} = (x_1, x_2, \ldots)$ be the input of the channel and $\boldsymbol{y} = (y_1, y_2, \ldots)$ the output of the channel. In this chapter, we will consider **discrete** channels, i.e., the channel inputs x_i take values on a finite alphabet $\mathcal{X}$ and the channel outputs y_i take values on a finite alphabet $\mathcal{Y}$. The communication channel depicted in Figure 4.1 is a discrete channel where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequences of bits, i.e., $\mathcal{X} = \mathcal{Y} = \{0, 1\}$. In general, the channel is noisy, hence the received sequence of bits $\boldsymbol{y}$ may be different to the transmitted sequence $\boldsymbol{x}$.

A (discrete) communication channel is completely specified by the conditional PMF $P_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x})$, i.e., the probability of receiving a the sequence of symbols $\boldsymbol{y}$ when the sequence of symbols $\boldsymbol{x}$ is transmitted. In this course, we will consider **memoryless** channels, for which the output of the channel only depends on the current input and not on the past inputs. Formally, a channel is memoryless if

$$\boxed{P_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x}) = \prod_i P_{Y|X}(y_i|x_i).}$$

Thus, a memoryless channel is completely specified by the conditional PMF $P_{Y|X}(y|x)$, i.e., the probability of receiving a (single) symbol y at the output of the channel when (a single) symbol x is transmitted. Therefore, formally speaking, a memoryless communication channel can be described as a stochastic process defined by the conditional PMF $P_{Y|X}(y|x)$.

Definition 4.1 (Discrete memoryless channel) A discrete memoryless channel is characterized by a discrete input alphabet $\mathcal{X}$, a discrete output alphabet $\mathcal{Y}$, and a conditional PMF $P_{Y|X}(y|x)$ which specifies the probability of observing $Y = y$ if $X = x$ was transmitted.

Hence, the channel input is a random variable X that takes values in $\mathcal{X}$ and the channel output is another random variable Y that takes values in $\mathcal{Y}$. The channel is entirely specified by $P(y|x)$.[1] The abstraction of a memoryless communication channel is depicted in Figure 4.2.

4.2 System Entropies

We learnt in Chapter 2 that the information content of a random variable, its **uncertainty**, is given by the entropy. Since in a communication system we want to *learn* about a transmitted random variable from the received one, it seems natural to consider the entropies of the random variables involved in the system.

We can now make use of the entropy defined in Chapter 2. We know that the average amount of

[1] We recall that we sometimes write $P(y|x)$ as a shorthand notation for $P_{Y|X}(Y = y|X = x)$.

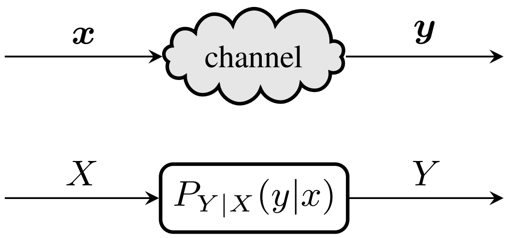


Figure 4.2: Communication channel with a sequence of symbols $\boldsymbol{x}$ at its input and a sequence of symbols $\boldsymbol{y}$ at its output and its mathematical representation.

uncertainty on the input of the channel X is given by

$$\mathsf{H}(X) = \sum_{x \in \mathcal{X}} P(x) \log \frac{1}{P(x)}. \tag{4.1}$$

We have shown in Section 2.2 (Lemma 2.1) that $0 \le \mathsf{H}(X) \le \log |\mathcal{X}|$ and that the entropy is 0 when the input X is known (in other words, it is deterministic). Furthermore, the entropy is maximum (the uncertainty about X is maximum) when all symbols $x \in \mathcal{X}$ are equally likely.

We can also write the average amount of uncertainty on the channel output Y as

$$\mathsf{H}(Y) = \sum_{y \in \mathcal{Y}} P(y) \log \frac{1}{P(y)}. \tag{4.2}$$

Similar to the conditional and joint probability for the case of two or more random variables, when we deal with more than one random variable we can consider the conditional and joint entropy.

We consider first the conditional entropy of random variables X given the event $Y = y$.

Definition 4.2 The conditional entropy of random variable X given the event $Y = y$ is defined as

$$\boxed{\mathsf{H}(X|Y = y) \triangleq \sum_{x \in \mathcal{X}} P(x|y) \log \frac{1}{P(x|y)}} \tag{4.3}$$

We are now ready to define the conditional entropy of X given the random variable Y, i.e., the uncertainty remaining in Y given that X is known.

Definition 4.3 (Conditional entropy) The conditional entropy of random variable X given the random variable Y

$$\begin{aligned}
\mathsf{H}(X|Y) &\triangleq \sum_{y \in \mathcal{Y}} P(y) \mathsf{H}(X|Y = y) \\
&= \sum_{y \in \mathcal{Y}} P(y) \sum_{x \in \mathcal{X}} P(x|y) \log \frac{1}{P(x|y)} \\
&= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(y) P(x|y) \log \frac{1}{P(x|y)} \\
&\overset{(a)}{=} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x,y) \log \frac{1}{P(x|y)}.
\end{aligned}$$

where in (a) we used Bayes' rule $P(x,y) = P(x|y)P(y)$.

We have the following very important theorem.

Theorem 4.1 (Conditioning reduces entropy) *For any two random variables X and Y,*

$$\boxed{\mathsf{H}(X|Y) \leq \mathsf{H}(X).}$$

with equality if and only if X and Y are statistically independent.

The joint entropy of two random variables X and Y is defined as follows.

Definition 4.4 (Joint entropy) The joint entropy of random variables X and Y is defined as

$$\mathsf{H}(X,Y) \triangleq \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P(x,y) \log \frac{1}{P(x,y)}. \tag{4.4}$$

If X and Y correspond to the input and output of the communication channel, the joint entropy gives the uncertainty of the overall system.

We have the following lemma.[2]

Lemma 4.1 *The joint entropy of two statistically independent random variables X and Y is the sum of entropies, i.e.,*

$$\boxed{\mathsf{H}(X,Y) = \mathsf{H}(X) + \mathsf{H}(Y).} \tag{4.5}$$

This result is expected: If two random variables are independent, then the joint uncertainty must be the sum of the uncertainties of each random variable separately.

Note that for any *useful* communication channel, the input X and the output Y **are not independent**. In this case, the joint entropy is given by the following lemma.

Theorem 4.2 (Chain rule) *Let X and Y be two discrete random variables. Then,*

$$\boxed{\mathsf{H}(X,Y) = \mathsf{H}(X) + \mathsf{H}(Y|X).} \tag{4.6}$$

Note that if X and Y are independent, X does not provide any information about Y, therefore $\mathsf{H}(Y|X) = \mathsf{H}(Y)$, and we obtain (4.5).

Proof: The result follows directly from $P(x,y) = P(x|y)P(y)$. ■

Note that, since $P(x,y) = P(y,x)$, it also follows that

$$\boxed{\mathsf{H}(X,Y) = \mathsf{H}(Y) + \mathsf{H}(X|Y).}$$

4.3 Information Conveyed by the Channel

We are now ready to consider how much information can be communicated over the channel. From the received symbol y we would like to determine the transmitted symbol x. In other words, at the receiver side we wish to determine how much information about the transmitted random variable X we gain from Y. We can ask ourselves a related question: **How much uncertainty remains about the knowledge of X after receiving Y?** This is precisely the **conditional entropy of X given Y**, $\mathsf{H}(X|Y)$. If Y determines X perfectly, obviously no uncertainty will remain on X, i.e., $\mathsf{H}(X|Y) = 0$, and we know which

[2] The proof of the lemma is trival by using $P(x,y) = P(x)P(y)$ and is left as an exercise.

symbol x was transmitted. If, on the contrary, Y provides no information about X, i.e., the channel is useless, $\mathsf{H}(X|Y) = \mathsf{H}(X)$.

Now, **how much information have we conveyed over the channel?** We can easily answer this question using the entropies of the system introduced above. *Before* reception of Y, the uncertainty on the transmitted random variable X is given by its entropy, $\mathsf{H}(X)$. *After* receiving Y, the uncertainty that remains on X is the conditional entropy $\mathsf{H}(X|Y)$. Thus, **the information communicated over the channel is the difference between these uncertainties**, a quantity that is referred to as **mutual information**.

Definition 4.5 (Mutual Information) The mutual information between X and Y is

$$\mathsf{I}(X;Y) \triangleq \mathsf{H}(X) - \mathsf{H}(X|Y). \tag{4.7}$$

The mutual information measures the **average reduction in uncertainty** about X that results from learning the value of Y.

From (4.7), by simple manipulation of the entropies, the mutual information can be written as

$$\boxed{\mathsf{I}(X;Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}.} \tag{4.8}$$

It is easy to see that the mutual information satisfies the following properties.

Theorem 4.3 *The mutual information satisfies the following properties:*

P1 $\mathsf{I}(X;Y) \geq 0$,

P2 $\mathsf{I}(X;Y) = 0$ *if and only if X and Y are independent,*

P3 $\mathsf{I}(X;Y) = \mathsf{I}(Y;X)$,

P4 $\mathsf{I}(X;Y) \leq \min(\mathsf{H}(X), \mathsf{H}(Y))$.

The proof of this lemma is left as an exercise (Hint: expand $\mathsf{I}(X;Y)$ using the entropies). However, we give here an intuitive explanation: (1) Obviously, the information that we can gain from X by knowing Y cannot be negative. At worst, Y gives no information about X, i.e., $\mathsf{I}(X;Y) \geq 0$. (2) If X and Y are independent, Y does not provide any information about X, hence $\mathsf{I}(X;Y) = 0$. If they are not independent, then Y provides some (perhaps very small) information about X. (3) Finally, the third property tells us that Y provides as much information about X than Y about X, which is intuitive. For this reason $\mathsf{I}(X;Y)$ is called *mutual* information.

A graphical description of the relations between the mutual information and the system entropies is given in Figure 4.3.

4.4 The Channel Capacity

We can now turn back to the initial question we posed in this chapter. Given a channel (defined by the conditional distribution $P(y|x)$), how much information can we transmit through the channel? We have no control over the channel. However, we do have in general control over the input distribution $P(x)$. The mutual information between the input and the output of the channel depends indeed on the chosen input distribution! We define the **maximum of the mutual information, maximized over all possible input distributions**, as the **channel capacity**.

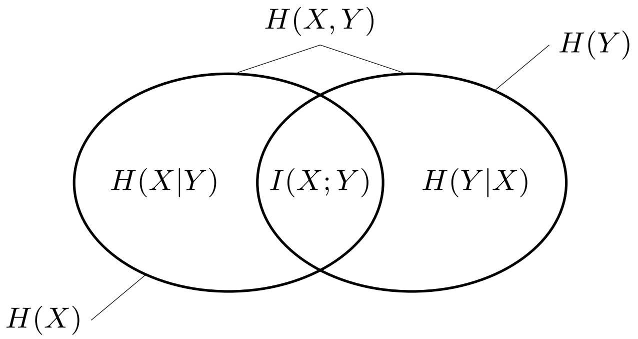


Figure 4.3: A graphical representation of the relations between entropy, conditional entropy, joint entropy and mutual information.

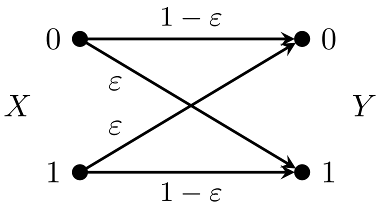


Figure 4.4: The binary symmetric channel with crossover probability ε.

Definition 4.6 (Channel capacity) For a given channel, the channel capacity is defined to be the maximum of the mutual information, maximized over all possible input distributions P_X,

$$\boxed{\mathsf{C} \triangleq \max_{P_X} \mathsf{I}(X;Y).} \tag{4.9}$$

4.5 An Example: Capacity of the Binary Symmetric Channel

We consider a very simple channel, the binary symmetric channel (BSC), with bits at the input and at the output. A bit is received correctly with probability $1-\varepsilon$ and is flipped with probability ε. The BSC is depicted in Figure 4.4.

The input and output of the BSC are binary random variables X and Y which take values on $\mathcal{X} = \{0,1\}$ and $\mathcal{Y} = \{0,1\}$, respectively. The channel is defined by the transition probabilities $P(y|x)$,

$$\boxed{\begin{aligned} P(0|0) &= P(1|1) = 1-\varepsilon \\ P(1|0) &= P(0|1) = \varepsilon. \end{aligned}}$$

Since $P(0|0) = P(1|1)$ and $P(1|0) = P(0|1)$, the channel is said to be symmetric.

In spite of its simplicity, the BSC is an important channel model with practical use. For instance, for an AWGN channel and BPSK transmission, if hard detection is performed at the output of the channel, the resulting discrete memoryless channel between the encoder and the decoder can be modeled as a BSC (we will discuss this in Section 7.4.1).[3]

We wish to compute the capacity of the BSC. First, we need to compute the mutual information between

[3] In fiber-optic communication systems, it is common to employ a binary code that is decoded using hard decision decoding to reduce dramatically the decoding complexity. In this case, the channel can be modeled as a BSC.

X and Y,

$$\mathsf{I}(X;Y) = \mathsf{H}(Y) - \mathsf{H}(Y|X).$$

We proceed by showing that

$$\mathsf{H}(Y|X=0) = \mathsf{H}(Y|X=1) = \mathsf{H}_\mathsf{b}(\varepsilon),$$

where $\mathsf{H}_\mathsf{b}(\cdot)$ is the binary entropy function introduced in Definition 2.3 (Chapter 2).

$$\begin{aligned}\mathsf{H}(Y|X=0) &= \sum_{y\in\mathcal{Y}} P(y|0)\log\frac{1}{P(y|0)}\\ &= P(1|0)\log\frac{1}{P(1|0)} + P(0|0)\log\frac{1}{P(0|0)}\\ &= \varepsilon\log\frac{1}{\varepsilon} + (1-\varepsilon)\log\frac{1}{(1-\varepsilon)} = \mathsf{H}_\mathsf{b}(\varepsilon).\end{aligned}$$

Similarly,

$$\begin{aligned}\mathsf{H}(Y|X=1) &= \sum_{y\in\mathcal{Y}} P(y|1)\log\frac{1}{P(y|1)}\\ &= P(0|1)\log\frac{1}{P(0|1)} + P(1|1)\log\frac{1}{P(1|1)}\\ &= \varepsilon\log\frac{1}{\varepsilon} + (1-\varepsilon)\log\frac{1}{(1-\varepsilon)} = \mathsf{H}_\mathsf{b}(\varepsilon).\end{aligned}$$

Now,

$$\begin{aligned}\mathsf{I}(X;Y) &= \mathsf{H}(Y) - \mathsf{H}(Y|X)\\ &= \mathsf{H}(Y) - \sum_{x\in\mathcal{X}} P(x)\mathsf{H}(Y|X=x)\\ &= \mathsf{H}(Y) - (P(0)\mathsf{H}(Y|X=0) + P(1)\mathsf{H}(Y|X=1))\\ &= \mathsf{H}(Y) - \underbrace{(P(0)+P(1))}_{=1}\mathsf{H}_\mathsf{b}(\varepsilon)\\ &= \mathsf{H}(Y) - \mathsf{H}_\mathsf{b}(\varepsilon) \qquad (4.10)\\ &\le 1 - \mathsf{H}_\mathsf{b}(\varepsilon), \qquad (4.11)\end{aligned}$$

where the last inequality follows since Y is a binary random variable, hence $\mathsf{H}(Y) \le \log|\mathcal{Y}| = \log 2 = 1$ (see Lemma 2.1 in Chapter 2).

Note that $\mathsf{I}(X;Y)$ depends on the probability mass function of Y (through $\mathsf{H}(Y)$), which, in turn, depends on the input probability mass function $P(x)$. The equality in (4.11) is attained if $\mathsf{H}(Y) = 1$, i.e., Y is uniform (the entropy $\mathsf{H}(Y)$ is maximum if $Y = 0$ and $Y = 1$ are equiprobable). This is achieved, in turn, if the input X is uniform. Thus, the capacity of the BSC channel is given by

$$\mathsf{C}_\mathsf{BSC} = 1 - \mathsf{H}_\mathsf{b}(\varepsilon) \quad \text{bits}, \qquad (4.12)$$

which is achieved with the input distribution $P(X=0) = P(X=1) = \frac{1}{2}$.

The mutual information of the BSC, $\mathsf{I}(X;Y) = \mathsf{H}(Y) - \mathsf{H}_\mathsf{b}(\varepsilon)$, (see (4.10)) is depicted in Figure 4.5 as a function of $P(X=0) = \delta$ (thus $P(X=1) = 1-\delta$), for several values of the channel crossover probability ε. It

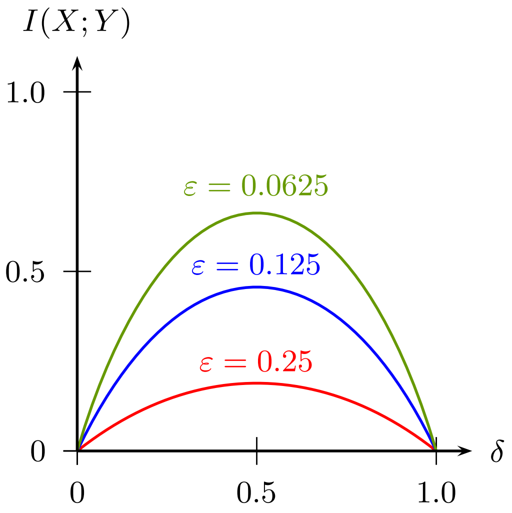


Figure 4.5: Mutual information of the binary symmetric channel (given in (4.10)) as a function of δ for three different values of the crossover probability ε.

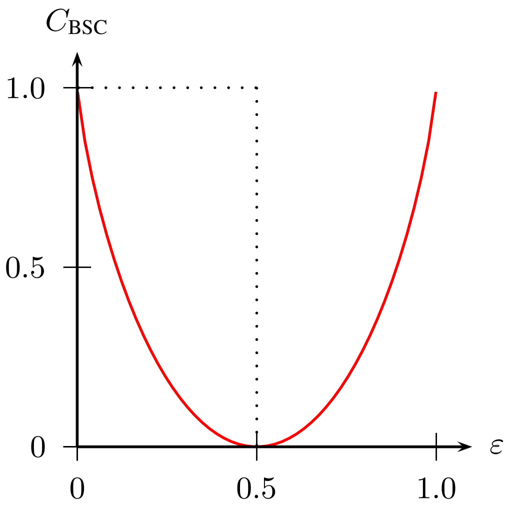


Figure 4.6: Capacity of the binary symmetric channel as a function of the crossover probability ε.

can be observed that in all cases the maximum of the mutual information is achieved for $\delta = \frac{1}{2}$ (equally likely symbols). Note also that the mutual information is smaller for higher values of ε. This result is expected: The more noisy the channel is, the less information we can communicate through it.

In Figure 4.6, we plot the capacity of the BSC versus the channel crossover probability. We observe that it achieves the maximum (1 bit per channel use) for $\varepsilon = 0$ (i.e., the channel is noiseless). In this case, we can transmit the bits directly over the channel and they will always be received correctly. Note that the capacity is also maximum for $\varepsilon = 1$. In this case, we can indeed also recover the transmitted bit with no error: it suffices to decide for the opposite bit to the one received. The minimum (zero) is achieved for $\varepsilon = \frac{1}{2}$. In this case, the channel output is independent of the channel input, therefore the channel cannot convey any information (the channel is useless!).

4.6 The Channel Coding Theorem

In the previous sections we have defined the channel capacity as the maximum of the mutual information. However, what is its operational meaning? Shannon showed that the channel capacity measures indeed the maximum transmission rate at which we can **communicate reliably** over the channel. We are now ready to state Shannon's channel coding theorem, one of the fundamental results in information theory.

Theorem 4.4 (Shannon's channel coding theorem) *For a discrete-time channel, it is possible to transmit information with an arbitrarily small probability of error if the communication rate R is below the channel capacity, i.e., $R < \mathsf{C}$. More precisely, for any $R < \mathsf{C}$, there exist a sequence of coding schemes of length N with average error probability $P_{\mathsf{e}}^{(N)}$ that tends to zero as $N \to \infty$, i.e., $P_{\mathsf{e}}^{(N)} \to 0$ as $N \to \infty$.*

Conversely, any sequence of coding schemes with vanishing error probability must have $R \leq C$. Hence, the probability of error for transmission above capacity is bounded above zero.

We say that all rates $R < \mathsf{C}$ are **achievable**.

Therefore, Shannon not only established the fundamental limit of reliable communication, the channel capacity, but also showed how to achieve it: by the use of **coding**. It is important to remark that here coding has a different meaning than (source) coding for data compression. The principle of **channel coding** (usually referred to as **error correcting coding** or simply as coding) is in fact the opposite than that of source coding: rather than compressing the data, channel coding introduces redundancy to the user information such that it can be exploited by the receiver to correct transmission errors. Shannon's channel coding theorem proves that, as long as the communication rate is below the channel capacity, $R < \mathsf{C}$, we can construct a code such that information can be communicated over the noisy channel with arbitrarily small error probability in the limit of infinitely large block lengths.

For instance, the capacity of the BSC for $\varepsilon = 0.25$ is $\mathsf{C}_{\mathsf{BSC}} = 1 - \mathsf{H}_{\mathsf{b}}(\varepsilon) = 1 - \mathsf{H}_{\mathsf{b}}(0.25) = 0.1887$ (see Figure 4.6). Thus, reliable communication over this channel is possible as long as we transmit at a rate smaller than 0.1887 bits per channel use, by using a properly designed code. Channel coding will be discussed in detail in Chapters 7–11.

4.7 Further Discussion

We should remark that, in general, finding a closed-form expression of the capacity for a given channel and the associated capacity-achieving input distribution is a difficult task.

On the other hand, it is important to highlight that Shannon proved the channel coding theorem by using random coding arguments, i.e., he proved the **existence of good codes** that achieve the channel capacity. However, he did not provide any clue on how to construct practical codes to achieve it. Therefore, since 1948, coding theorists have been pursuing the quest of searching for practical codes (i.e., codes that can be decoded with reasonable complexity) that achieve the promised limits.

Chapter 5

Communication over the AWGN Channel

In the previous chapter we have established the fundamental limit of reliable communication over a channel, the **channel capacity**. In this chapter, we specialize the results for the AWGN channel. Due to its prevalence and widespread use for both analysis and modeling, the AWGN channel is a very important channel model.

Our focus here is on the (complex) AWGN channel and we assume that the channel is **memoryless**. We start with the continuous-time AWGN channel,

$$y(t) = x(t) + n(t),$$

where $x(t)$ is bandlimited with bandwidth W and has signal power P, the symbol interval is $T = 1/W$ (or equivalently the symbol rate is W (complex) symbols/s), and $n(t)$ is complex AWGN with two-sided PSD N_0. The signal-to-noise ratio (SNR) is

$$\boxed{\mathsf{SNR} \triangleq \frac{P}{\mathsf{N}_0 W}.} \tag{5.1}$$

We also consider the corresponding discrete-time AWGN channel,

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n},$$

where $\boldsymbol{x} = (x_1, x_2, \ldots)$ is the input of the channel, i.e., the transmitted sequence of constellation symbols (for example symbols from a QAM constellation) with average energy per symbol $\mathsf{E_s}$, $\boldsymbol{y} = (y_1, y_2, \ldots)$ is the received sequence, and $\boldsymbol{n}$ is a sequence of i.i.d. Gaussian noise random variables with zero mean and variance $\sigma^2 = \mathsf{N}_0/2$ per real dimension. A (simplified) block diagram of a communication system is depicted in Figure 5.1. The source data $\boldsymbol{u}$ at the output of the source encoder is first encoded by an error correcting code into codeword $\boldsymbol{c}$, which is then modulated into a sequence of constellation symbols $\boldsymbol{x}$ that are transmitted over the AWGN channel. The received sequence $\boldsymbol{y}$ is demodulated and fed to the decoder, which aims at recovering the source data. The modulator and demodulator are denoted by Φ and Φ^{-1}, respectively, in the figure.

If the channel is memoryless (see Section 4.1), it is completely specified by the conditional PDF $p_{Y|X}(y|x)$ of receiving y if symbol x was transmitted. The input of the channel is a random variable X that takes values on the finite alphabet $\mathcal{X} = \{\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_M\} \subset \mathbb{C}$ according to a PMF $P_X(x)$. Note that $\mathcal{X}$ is the constellation, of cardinality M, imposed by the modulation. The output of the channel is a Gaussian-distributed random

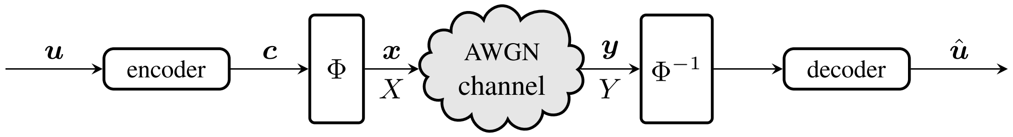


Figure 5.1: Simplified block diagram of a communication system. The input of the channel is a sequence of constellation symbols $x_i \in \mathcal{X}$.

variable Y. The PDF $p_{Y|X}(y|x)$ is often referred to as the **channel law**. For the AWGN channel, the channel law reads

$$\boxed{p_{Y|X}(y|x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\|y-x\|^2}{2\sigma^2}}.} \tag{5.2}$$

The SNR of the discrete-time channel is defined as

$$\boxed{\mathsf{SNR} \triangleq \frac{\mathsf{E_s}}{\mathsf{N_0}} = \frac{\mathsf{E_s}}{2\sigma^2}.} \tag{5.3}$$

Note that the SNR of the discrete-time AWGN channel (5.3) is the same as that of the continuous-time AWGN channel (5.1). In fact, since $P = \mathsf{E_s}/T$, i.e., $P = \mathsf{E_s}W$, we have

$$\boxed{\mathsf{SNR} \triangleq \frac{P}{\mathsf{N_0}W} = \frac{\mathsf{E_s}}{\mathsf{N_0}} = \frac{\mathsf{E_s}}{2\sigma^2}.}$$

As shown in the previous chapter, for a given distribution P_X of the channel input, the amount of information that can be conveyed over the channel is given by the mutual information $\mathsf{I}(X;Y)$ (see (4.8)),

$$\boxed{\mathsf{I}(X;Y) = \iint p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy,}$$

where $p(x,y) = p(y|x)p(x)$ and the double summation with respect to x and y in (4.8) becomes a double integral since the random variable Y at the output of the channel is continuous and, in principle, X can also be continuous. Note that for the case of transmission using a finite number of signal points, e.g., PAM or QAM modulation, X is a discrete random variable and the mutual information is

$$\boxed{\mathsf{I}(X;Y) = \sum_{x\in\mathcal{X}} \int p(x,y) \log \frac{p(x,y)}{P(x)p(y)} dy,} \tag{5.4}$$

where $p(x,y) = p(y|x)P(x)$.

We have also seen in the previous chapter that the ultimate limit at which we can transmit reliably is given by the channel capacity, obtained by maximizing $\mathsf{I}(X;Y)$ over all possible input distributions,

$$\mathsf{C} \triangleq \max_{p_X} \mathsf{I}(X;Y). \tag{5.5}$$

Here, we will not derive the capacity of the AWGN channel but we give its value in the following two propositions.

Proposition 5.1 (Capacity of the continuous-time AWGN channel) *The channel capacity of the continuous-time AWGN channel of bandwidth W is*

$$\mathsf{C}_{\mathsf{AWGN-C}} = W \log(1 + \mathsf{SNR}) \;\; \text{[bits/s]},$$

where the SNR is defined in (5.1).

The channel capacity is achieved by a Gaussian input distribution, i.e., X is Gaussian-distributed.

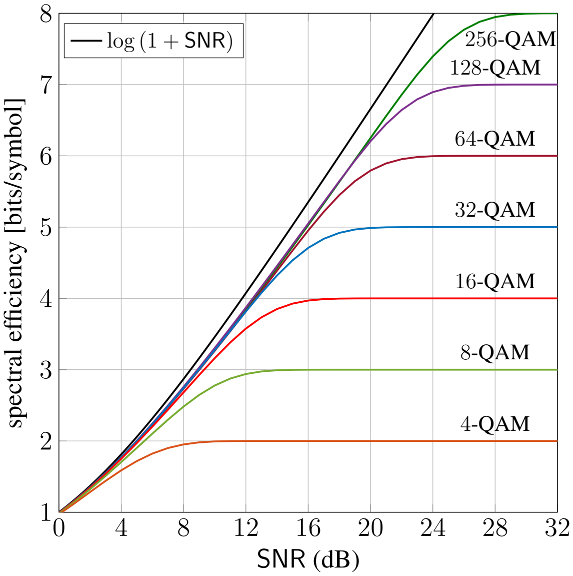


Figure 5.2: Channel capacity of the AWGN channel and mutual information curves for several QAM constellations.

Note that for a bandlimited AWGN channel, the capacity depends on only two parameters, the **bandwidth W and the SNR**.

Proposition 5.2 (Capacity of the discrete-time AWGN channel) *The channel capacity of the discrete-time real AWGN channel with average energy per symbol $\mathsf{E_s}$ and noise variance $\sigma^2 = \mathsf{N_0}/2$ is*

$$\mathsf{C_{AWGN-D}} = \frac{1}{2}\log\left(1 + \frac{\mathsf{E_s}}{\sigma^2}\right) \text{[bits/channel use] or [bits/symbol]}, \tag{5.6}$$

and is achieved by a Gaussian input distribution, i.e., $X \sim \mathcal{N}(0, \mathsf{E_s})$.

The channel capacity of the discrete-time complex AWGN channel with average energy per symbol $\mathsf{E_s}$ and noise variance $\sigma^2 = \mathsf{N_0}/2$ per dimension is

$$\mathsf{C_{AWGN-D}} = \log\left(1 + \frac{\mathsf{E_s}}{2\sigma^2}\right) \text{[bits/channel use] or [bits/symbol]}, \tag{5.7}$$

and is achieved by a Gaussian input distribution, i.e., $X \sim \mathcal{CN}(0, \mathsf{E_s})$.

The capacity of the discrete-time AWGN channel (5.7) is depicted in Figure 5.2. In the figure, we also plot the mutual information $\mathsf{I}(X;Y)$ in (5.4) for several QAM constellations with uniform signaling, i.e., all constellation symbols are transmitted with the same probability. Note that the mutual information for an M-QAM constellation flattens out for high SNRs to $\log M$ bits per symbol (with an M-QAM constellation we cannot transmit more than $\log M$ bits per symbol!). Note also that there is a gap between the mutual information curves and the capacity. The reason is that QAM constellations with uniform signaling are not capacity achieving! (the capacity-achieving distribution is a Gaussian distribution).

It can be easily shown that the capacities of the continuous-time and discrete-time AWGN channels are equivalent. To show that, we focus on the complex AWGN channel,

$$\mathsf{C_{AWGN-D}} = \log\left(1 + \frac{\mathsf{E_s}}{2\sigma^2}\right) = \log(1 + \mathsf{SNR}) \text{[bits/channel use] or [bits/symbol]}. \tag{5.8}$$

Now, dividing (5.8) by the symbol interval T, we obtain

$$\mathsf{C}_{\mathsf{AWGN-C}} = \frac{1}{T}\log(1+\mathsf{SNR}) \text{ [bits/second]}.$$

Finally, setting $T = \frac{1}{W}$ we get

$$\mathsf{C}_{\mathsf{AWGN-C}} = W\log(1+\mathsf{SNR}) \text{ [bits/second]},$$

which is the capacity of the continuous-time channel.

5.1 The Channel Coding Theorem for the AWGN Channel

As discussed in the previous chapter, Shannon not only established that there is a maximum at which information can be transmitted reliably over a channel, but he also proved that the capacity can be achieved by the use of coding. We can now restate the channel coding theorem for the particular case of the AWGN channel.

Theorem 5.1 (The channel coding theorem for the AWGN channel) *Consider a discrete-time AWGN channel with capacity $\mathsf{C}_{\mathsf{AWGN-D}}$ given in Proposition 5.2. Then, all rates R below $\mathsf{C}_{\mathsf{AWGN-D}}$ are achievable, i.e., for every $R < \mathsf{C}_{\mathsf{AWGN-D}}$ there exists a sequence of coding schemes with vanishing error probability $P_{\mathsf{e}}^{(N)} \to 0$ as the block length $N \to \infty$. Conversely, any sequence of coding schemes of rate R and block length N with error probability $P_e^N \to 0$ must have a rate $R \leq \mathsf{C}_{\mathsf{AWGN-D}}$.*

The theorem tells us that arbitrarily reliable transmission can be achieved over the discrete-time AWGN channel at any rate $R < \mathsf{C}_{\mathsf{AWGN-D}}$ if codes of large length are used. We then say that the rate R is **achievable**. With reference to Figure 5.2, for a given SNR, all rates R below the capacity curve $\log(1+\mathsf{SNR})$ are achievable. Conversely, we cannot transmit with rate R above the black curve with vanishing error probability. Note that if we transmit with a given constellation, say 16-QAM, then the reference curve is that of the corresponding mutual information. In this case, all rates below the red curve are achievable.

Correspondingly, for the continuous-time channel reliable transmission is possible at any bit rate R_{b} (in [bits/second]) such that $R_{\mathsf{b}} < \mathsf{C}_{\mathsf{AWGN-C}}$, where $R_{\mathsf{b}} = R/T$, since we transmit one symbol each T seconds.

5.2 Power Efficiency and Bandwidth Fundamental Tradeoff

Shannon's channel coding theorem is very useful to understand the fundamental tradeoffs underlying a communication system. To compare the performance of coded communication systems, rather than considering the SNR $\mathsf{E_s}/\mathsf{N_0}$, it is useful to use the ratio $\mathsf{E_b}/\mathsf{N_0}$ instead, where $\mathsf{E_b}$ denotes the **energy per information bit**. It is common to refer to $\mathsf{E_b}/\mathsf{N_0}$ as the **power efficiency**. Consider a coded system which encodes sequences $\boldsymbol{u}$ of K information bits onto sequences $\boldsymbol{x}$ of N symbols, i.e., the transmission rate is $R = K/N$ bits per symbol. Note that here by coded system we mean the combination of the channel code and the modulator, see Figure 5.1. If each symbol is transmitted with energy $\mathsf{E_s}$, the amount of energy conveyed by each information bit amounts

$$\boxed{\mathsf{E_b} = \frac{\mathsf{E_s}}{R}.}$$

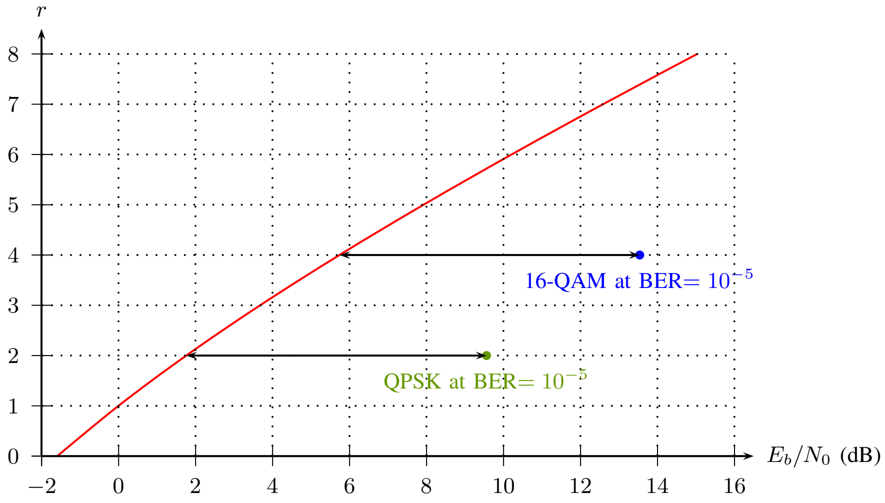


Figure 5.3: Minimum required $\mathsf{E_b}/\mathsf{N_0}$ for reliable communication for a given spectral efficiency R. Below $\mathsf{E_b}/\mathsf{N_0} = -1.59\,\text{dB}$ reliable communication is not possible.

We know that the transmission rate must satisfy $R < \mathsf{C}_{\mathsf{AWGN-D}}$ [bits/symbol]. Starting from (5.7), we can proceed as

$$\begin{aligned} R &< \mathsf{C}_{\mathsf{AWGN-D}} \\ &= \log\left(1 + \frac{\mathsf{E_s}}{2\sigma^2}\right) \\ &= \log\left(1 + R\frac{\mathsf{E_b}}{2\sigma^2}\right) \\ &= \log\left(1 + R\frac{\mathsf{E_b}}{\mathsf{N_0}}\right). \end{aligned}$$

Therefore, for reliable transmission, the minimum required $\mathsf{E_b}/\mathsf{N_0}$ to support a transmission rate R is given by

$$\boxed{\frac{\mathsf{E_b}}{\mathsf{N_0}} > \frac{2^R - 1}{R}.} \tag{5.9}$$

The right hand side of (5.9) is monotonically increasing with R and the minimum is achieved for $R \to 0$,

$$\lim_{R\to 0} \frac{2^R - 1}{R} \overset{(a)}{=} \lim_{R \to 0} \frac{2^R \ln 2}{1} = \ln 2,$$

where in (a) we used l'Hôpital's rule and the fact that $f'(a^x) = a^x \ln a$. Hence,

$$\boxed{\mathsf{E_b}/\mathsf{N_0} > \ln 2 = -1.59 \text{ dB},} \tag{5.10}$$

i.e., it is not possible to transmit reliably over the AWGN channel at $\mathsf{E_b}/\mathsf{N_0}$ smaller than −1.59 dB, even when we let $R \to 0$.

We plot (5.9) in Figure 5.3. All rates R below the red curve are achievable. Equivalently, for a given transmission rate R it is possible to transmit with vanishing error probability for values of $\mathsf{E_b}/\mathsf{N_0}$ in the right of the red curve.

The transmission rate R (with units [bits/symbol] or [bits/channel use]) is usually referred to as the **spectral efficiency**. In fact, note that R is directly related to the bandwidth of the continuous-time

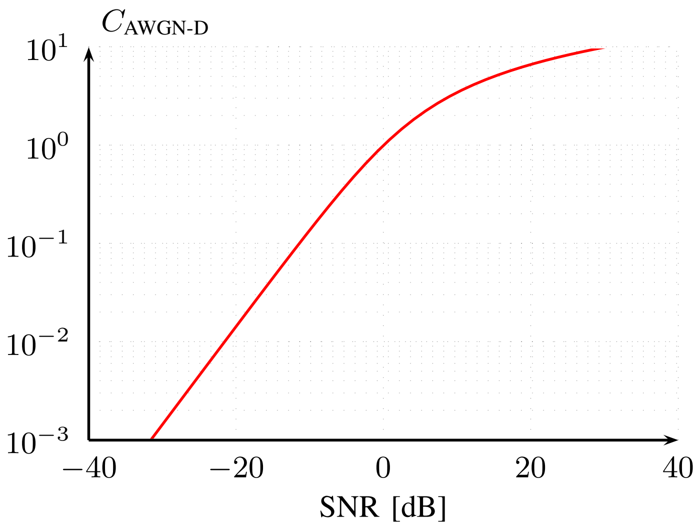


Figure 5.4: Capacity $\mathsf{C}_{\mathsf{AWGN-D}}$ as a function of SNR. For low SNR the gain is linear, whereas for high SNR the gain is only logarithmic.

channel,

$$\boxed{R = R_{\mathsf{b}}T = R_{\mathsf{b}}/W,} \tag{5.11}$$

i.e., R determines how efficiently we use the spectrum. From (5.11), to support the same information rate R_{b} (i.e., for fixed R_{b}), decreasing the spectral efficiency implies increasing the required bandwidth. Conversely, decreasing the used bandwidth implies increasing the spectral efficiency.

☒ Therefore, equation (5.9) and Figure 5.3 show a **fundamental tradeoff between power efficiency and bandwidth efficiency**: The required $\mathsf{E_b/N_0}$, and therefore the required power, increases with increasing values of the spectral efficiency R, while increasing R decreases the required bandwidth to support the same information rate R_{b}. On the other hand, decreasing R requires less power, but higher bandwidth to support the same R_{b}.

We can compare any practical communication scheme with the curve shown in Figure 5.3. For instance, both uncoded BPSK and 4-QAM correspond to $R = 2$ [bits/symbol] (for two dimensions) and require $\mathsf{E_b/N_0} \approx 9.5$ dB to achieve an error rate of 10^{-5}, which is about 7.8 dB away from the minimum required $\mathsf{E_b/N_0}$ for reliable transmission. In contrast, modern codes in combination with a higher-order constellation (for the same spectral efficiency $R = 2$) can achieve very low error rates at $\mathsf{E_b/N_0} \approx 1$ dB away from the theoretical limit.

5.3 Power-Limited and Band-Limited Regimes

Using $\mathsf{SNR} = \frac{\mathsf{E_s}}{2\sigma^2} = R\frac{\mathsf{E_b}}{\mathsf{N_0}}$, we can write (5.9) in terms of the SNR as

$$\boxed{\mathsf{SNR} > 2^R - 1.} \tag{5.12}$$

As before, the fundamental tradeoff between power and bandwidth shows up: Increasing R increases the required SNR, and hence the required power, while decreasing the required bandwidth to support the same information rate R_{b}, and vice versa.

Ideal band-limited AWGN channels may be classified as **bandwidth-limited** ($\mathsf{SNR} \gg 1$) or **power-limited** ($\mathsf{SNR} \ll 1$) according to whether they permit transmission at high spectral efficiencies or not. The behavior of the capacity is very different in the two regimes, as we will see in the following.

5.3.1 Power-Limited Regime

In the power-limited regime, the SNR is small, while W can grow very large. In this case, we can approximate the channel capacity as

$$\boxed{\begin{aligned} \mathsf{C}_{\mathsf{AWGN-D}} &= \log(1+\mathsf{SNR}) \\ &\approx \frac{1}{\ln 2}\cdot\mathsf{SNR}, \end{aligned}} \tag{5.13}$$

where in (5.13) we used $\ln(1+x) \approx x$ for small x and $\log a = \frac{\ln a}{\ln 2}$. In other words, in the power-limited regime, $\mathsf{C}_{\mathsf{AWGN-D}}$ increases **linearly** with the SNR (see Figure 5.4). From (5.13) it results that in the power-limited regime doubling the SNR doubles the capacity.

Since power is so precious, in the power-limited regime we commonly use BPSK (in this case $R = 2$ for two dimensions) at the expense of bandwidth efficiency. It can be seen in Figure 5.2 that BPSK requires a smaller SNR than higher-order constellations. However, since the spectral efficiency is low, a larger bandwidth is required. Some of the most common modulation formats will be discussed in Chapter 6.

Example 5.1 An example of power-limited communication is satellite communication. There are no bandwidth restrictions on a deep-space communication channel, therefore it makes sense to use as much bandwidth as possible, while the power (directly related to the life-time of the satellite) is limited.

5.3.2 Bandwidth-Limited Regime

In the bandwidth-limited regime, we can accept to transmit at high power, i.e., the SNR can grow very large. In this case,

$$\boxed{\begin{aligned} \mathsf{C}_{\mathsf{AWGN-D}} &= \log(1+\mathsf{SNR}) \\ &\approx \log\mathsf{SNR}, \end{aligned}}$$

so that $\mathsf{C}_{\mathsf{AWGN-D}}$ increases only **logarithmically** with the SNR (see again Figure 5.4).

In the bandwidth-limited regime, every doubling of the SNR (every additional 3 dB in SNR) yields an increase in $\mathsf{C}_{\mathsf{AWGN-D}}$ of only 1 bit per symbol. In the bandwidth-limited regime, we commonly use higher-order modulation at the expense of power (see Figure 5.2).

Chapter 6

Analysis of Linear Modulations

In Chapter 5 we have established the fundamental limits of reliable transmission. If no constraint is imposed on the channel input, the fundamental limit is the channel capacity. On the other hand, if the channel input is constrained to a given constellation, e.g., QAM, and a given $P_X(x)$, the limit is given by the corresponding mutual information. The channel capacity and the mutual information serve as a benchmark to compare the performance of actual communication systems. Ideally, we would like a communication system to perform as close as possible to capacity. In this chapter, we evaluate the performance of uncoded transmission over the AWGN channel using M-PSK and M-QAM modulation, two of the most popular modulation formats, used in many communication systems.

The evaluation of the performance of modulation formats is based on three parameters.

- The error probability (symbol error probability or bit error probability),
- The required $\mathsf{E_b}/\mathsf{N_0}$ to achieve such probability of error, and
- The spectral efficiency R.

The probability of error is a measure of the reliability of the transmission, the required $\mathsf{E_b}/\mathsf{N_0}$ is a measure of the power efficiency, and R is a measure of how efficiently the modulation format makes use of the bandwidth.

In this chapter, we will see that, non surprisingly, to achieve a low probability of error, uncoded transmission performs far away from capacity, i.e., it requires an $\mathsf{E_b}/\mathsf{N_0}$ much larger than the theoretical limit. Indeed, Shannon showed that to achieve capacity coding is needed! Analyzing the performance of uncoded transmission is nevertheless informative. First, we can realize how *poorly* the uncoded system performs. Moreover, for a given constellation, the difference between the performance achieved by the uncoded system and the corresponding mutual information curve determines the maximum possible gain that can be achieved using coding.

We will assume transmission over the AWGN channel,

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n},$$

where $\boldsymbol{x} = (x_1, x_2, \ldots)$ is the sequence of symbols transmitted over the channel with average energy per symbol $\mathsf{E_s}$, $\boldsymbol{y} = (y_1, y_2, \ldots)$ is the sequence of received values, andÂ $\boldsymbol{n} = (n_1, n_2, \ldots)$ is a sequence of realizations of a Gaussian process with zero mean and variance per real dimension $\sigma^2 = \mathsf{N_0}/2$, $N_i \sim \mathcal{N}_{\mathbb{C}}(0, 2\sigma^2)$. As

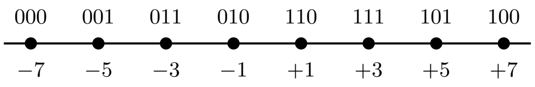


Figure 6.1: 8-PAM constellation with Gray labeling. The channel input X takes values on a set of 8 different symbols.

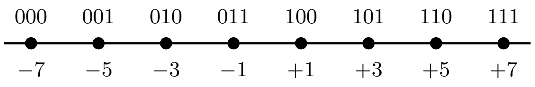


Figure 6.2: 8-PAM constellation with natural labeling. The channel input X takes values on a set of 8 different symbols.

we have discussed in the previous chapter, the input of the channel can be seen as a sequence of random variables $X_1, X_2 \ldots$ that take values on a finite set $\mathcal{X} = \{\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_M\} \subset \mathbb{C}$, i.e., $\mathcal{X}$ is the constellation of cardinality M imposed by the modulation. Note that for a constellation with M symbols, $m = \log M$ bits are mapped to each constellation symbol, i.e., we transmit $m = \log M$ bits per symbol.[1] Therefore, the energy per information bit is

$$\mathsf{E_b} = \frac{\mathsf{E_s}}{\log M}.$$

As an example, consider transmission using the 8-PAM constellation depicted in Figure 6.1. In this case, X is a random variable that takes values on the set $\mathcal{X} = \{-7, -5, -3, -1, +1, +3, +5, +7\}$ according to a given distribution P_X. For conventional constellations, all symbols are equiprobable, i.e., $P_X(\mathsf{X}_i) = 1/M$ for all X_i. A tuple of $m = \log 8 = 3$ bits is mapped to each constellation symbol by the modulator Φ. For example, the tuple 010 is mapped to constellation symbol -3. In the figure, we assumed Gray mapping, i.e., the binary labelings of two neighboring constellation symbols differ only by one bit. Obviously, the way bits are mapped to constellation symbols is not unique. In Figure 6.2, we plot an 8-PAM constellation with natural labeling.

The block diagram of the uncoded communication system is depicted in Figure 6.3. It is a simplified version of that of Figure 5.1 in which the encoder and the decoder have been removed.

6.1 Optimum Decoding Rule

With reference to Figure 6.3, based on the noisy observation $\boldsymbol{y}$, the goal of the receiver is to infer the transmitted sequence $\boldsymbol{u}$ optimally. With some abuse of language, we will refer to the process of inferring $\boldsymbol{u}$ from $\boldsymbol{y}$ as decoding. Note that inferring $\boldsymbol{u}$ from $\boldsymbol{y}$ is equivalent to inferring the transmitted sequence $\boldsymbol{x}$ from $\boldsymbol{y}$, since the mappings $\boldsymbol{u} \to \boldsymbol{x}$ is invertible, i.e., from $\boldsymbol{x}$ we can recover $\boldsymbol{u}$ univocally. Decoding is a probabilistic inference problem: the decoder estimates the most likely transmitted sequence $\boldsymbol{x}$ given the noisy channel output $\boldsymbol{y}$. The most natural decoding criterion is that of minimizing the probability of error. It can be shown that this is equivalent to maximizing $p(\boldsymbol{x}|\boldsymbol{y})$,[2] referred to as the **a posteriori probability**. The

[1] As before, unless otherwise stated, we will assume that the logarithm is in base 2.

[2] Note that here we use the conditional PDF $p(\boldsymbol{x}|\boldsymbol{y})$ since Y is typically a sequence of continuous random variables. This is the case for transmission over the AWGN channel.

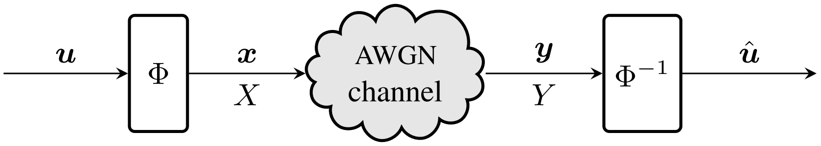


Figure 6.3: Uncoded communication system. The input of the channel is a sequence of constellation symbols $x_i \in \mathcal{X}$.

corresponding decoding rule is referred to as **maximum a posteriori** (MAP),

$$\hat{\boldsymbol{x}}_{\mathsf{MAP}} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}). \tag{6.1}$$

We can rewrite (6.1) using Bayes as

$$\begin{aligned}\hat{\boldsymbol{x}}_{\mathsf{MAP}} &= \arg\max_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y})\\ &= \arg\max_{\boldsymbol{x}} \frac{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})}\\ &\overset{(a)}{=} \arg\max_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}),\end{aligned} \tag{6.2}$$

where in (a) we used the fact that $p(\boldsymbol{y})$ is independent of $\boldsymbol{x}$.

Typically, we can assume that all sequences $\boldsymbol{x}$ are equiprobable. In this case,

$$\arg\max_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x}).$$

Using this in (6.2), the MAP decoding rule (6.1) boils down to

$$\hat{\boldsymbol{x}}_{\mathsf{ML}} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x}), \tag{6.3}$$

which is referred to as the **maximum likelihood** (ML) decoding rule.

If the channel is memoryless (the output of the channel y_i depends only on the input x_i and not on the past input symbols $x_1, \ldots, x_{i-1}$), without loss of optimality we can use a symbol-by-symbol decoder that makes an independent decision on each received symbol y_i. Assuming that all symbols of the constellation $\mathcal{X}$Â are transmitted with equal probability, the optimum decoding rule is the ML rule,

$$\hat{x}_i = \arg\max_{x_i \in \mathcal{X}} p(y_i|x_i), \tag{6.4}$$

where $y_i = x_i + n_i$, with $N_i \sim \mathcal{N}_{\mathbb{C}}(0, 2\sigma^2)$. From now on, for the ease of notation, and since it is unnecessary, we will remove the subindex i and simply write

$$\hat{x} = \arg\max_{x \in \mathcal{X}} p(y|x). \tag{6.5}$$

Let us now particularize (6.5) for transmission over the Gaussian channel, for which the channel law

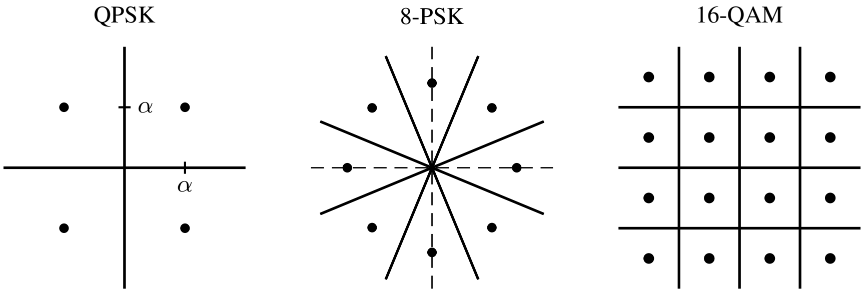


Figure 6.4: ML decision regions for 4-QAM, 8-PSK and 16-QAM.

$p(y|x)$ is given in (5.2) (see Chapter 5). We proceed from (6.5) as follows,

$$\begin{aligned}
\hat{x} &= \arg\max_{x\in\mathcal{X}} p(y|x)\\
&= \arg\max_{x\in\mathcal{X}} \left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\|y-x\|^2}{2\sigma^2}}\right)\\
&\overset{(a)}{=} \arg\max_{x\in\mathcal{X}} \ln\left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\|y-x\|^2}{2\sigma^2}}\right)\\
&= \arg\max_{x\in\mathcal{X}} \left(\ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right)-\frac{\|y-x\|^2}{2\sigma^2}\right)\\
&\overset{(b)}{=} \arg\max_{x\in\mathcal{X}} \left(-\frac{\|y-x\|^2}{2\sigma^2}\right)\\
&= \arg\max_{x\in\mathcal{X}} \left(-\|y-x\|^2\right)\\
&\overset{(c)}{=} \arg\min_{x\in\mathcal{X}} \|y-x\|^2,
\end{aligned}$$

where in (a) we used the fact that the logarithm is a monotonic function, in (b) we exploit the fact that addition of a constant does not change the maximization, and in (c) that maximizing a function is equivalent to minimizing minus the function.

Therefore, for transmission over the AWGN channel the ML decoding rule is

$$\begin{aligned}
\hat{x} &= \arg\min_{x\in\mathcal{X}} \|y-x\|^2 \qquad (6.6)\\
&= \arg\min_{x\in\mathcal{X}} d_{\mathsf{E}}^2(x,y)\\
&= \arg\min_{x\in\mathcal{X}} d_{\mathsf{E}}(x,y).
\end{aligned}$$

In other words, the ML decoding rule corresponds to selecting among all possible constellation symbols $x \in \mathcal{X}$ the one at minimum Euclidean distance to the received value y.

6.1.1 Maximum Likelihood Decision Regions

The minimum Euclidean distance criterion resulting from the ML rule can be geometrically interpreted in terms of ML decision regions. The ML decision region for a symbol $x \in \mathcal{X}$ is the geometric region corresponding to all the points that are closer to x than to any other symbol of the constellation. Formally, for two-dimensional constellations, the decision region for constellation symbol X_1, denoted by $\mathcal{R}_i$, is

$$\mathcal{R}_i = \{y \in \mathbb{C} : \|y-\mathsf{X}_i\|^2 \leq \|y-\mathsf{X}_j\|^2 \ \forall j \neq i\}. \qquad (6.7)$$

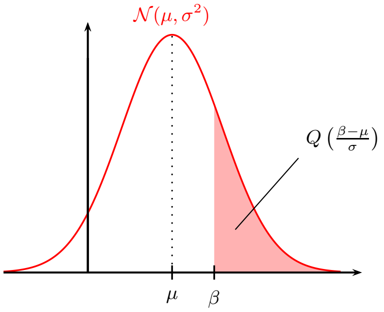


Figure 6.5: Use of the Q-function to compute the tail probability for a Gaussian-distributed random variable with mean μ and variance σ^2.

The minimum-distance regions $\mathcal{R}_i$ are also called *Voronoi regions*.

In Figure 6.4, we show the ML decision regions for 4-QAM, 8-PSK and 16-QAM constellations.

6.2 Preliminaries

Once discussed the optimum decoding rule, we would like to analyze the performance of different modulation formats under ML decoding. In this section, we introduce two mathematical tools that will be very useful for the analysis of modulation formats in the following sections.

6.2.1 The Union Bound and the Q-function

A useful tool to bound the error probability is the **union bound**. The union bound simply states the following: given a number of events $E_1, \dots, E_n$ (for example, for a deck of poker cards the following events can be defined, $E_1 =$"the card is red", $E_2 =$"the card is a spade", $E_3 =$ "the card has a value larger than 5", etc.), then the probability of the union of the events is upperbounded by the sum of probabilities, i.e.,

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \Pr(E_i), \tag{6.8}$$

where $E_i \cup E_j$ stands for the union of the events E_i and E_j. Observe that the right-hand-side of (6.8) can be greater than one (the sum of probabilities is not a probability!), in which case the union bound is not a useful bound. The union bound, however, is very useful to gain insight into the behavior of the error probability for high $\mathsf{E_b}/\mathsf{N_0}$.

A second important tool is the so-called **Q-function**. Let Z be a Gaussian random variable with mean μ and variance σ^2, $Z \sim \mathcal{N}(\mu, \sigma^2)$. Then, for a given constant β,

$$\Pr(Z > \beta) = \mathsf{Q}\left(\frac{\beta - \mu}{\sigma}\right), \tag{6.9}$$

where $\mathsf{Q}(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\tau^2/2\right) \mathrm{d}\tau$ is the tail probability of the standard normal distribution. The probability $\Pr(Z > \beta)$ is depicted in Figure 6.5 (the red-colored area below the curve).

The Q-function has the property

$$\mathsf{Q}(-x) = 1 - \mathsf{Q}(x). \tag{6.10}$$

While there is no closed-form expression for the Q-function, we have the following bound for $x > 0$,

$$\boxed{\mathsf{Q}(x) < \frac{1}{2} e^{-x^2/2}.} \tag{6.11}$$

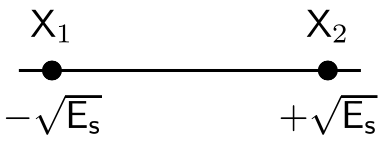


Figure 6.6: Binary phase shift keying constellation.

Note that the Q-function decreases with its argument x. Furthermore, the Q-function has the following important property: for $0 < a \ll b < c$

$$\mathsf{Q}(a) + \mathsf{Q}(b) + \mathsf{Q}(c) \approx \mathsf{Q}(a), \tag{6.12}$$

which often simplifies expressions.

We are now ready to evaluate the probability of error of the most well-known modulation formats. We will first consider simple BPSK modulation and then higher-order modulation with $M > 2$ signal points.

6.3 Symbol Error Probability of Linear Modulation Formats

In this section, we analyze the symbol error probability of several linear modulation formats. In particular, we derive the exact symbol error probability for BPSK and 4-QAM modulations. For higher-order modulations, deriving the exact symbol error probability is difficult. In this case, we derive an upper bound as well as a simple approximation to the symbol error probability.

6.3.1 Symbol Error Probability of Binary Phase Shift Keying

For BPSK, $\mathcal{X} = \{\mathsf{X}_1, \mathsf{X}_2\} \subset \mathbb{R}$, where $\mathsf{X}_1 = -\sqrt{\mathsf{E_s}}$ and $\mathsf{X}_2 = \sqrt{\mathsf{E_s}}$ such that the average energy per symbol is $\mathsf{E_s}$. The constellation is drawn in Figure 6.6.

The received signal y is also real,

$$y = x + n,$$

where $N \sim \mathcal{N}(0, \sigma^2)$. When the two constellation symbols X_1 and X_2 are equiprobable, the optimal decision rule is the ML rule (see (6.3) and (6.6)), i.e., select among X_1 and X_2 the symbol that is closer in Euclidean distance to the received value y. Thus, $\hat{x} = \mathsf{X}_2$ if $y > 0$ and $\hat{x} = \mathsf{X}_1$ otherwise.

We compute in the following the probability that the ML decoder decides on the wrong symbol, i.e., the symbol error probability.[3]

$$\begin{aligned} P_{\mathsf{s}}^{\mathsf{BPSK}} &= \sum_{x \in \mathcal{X}} \Pr(\hat{x} \neq x | x) P(x) \\ &= \Pr(\mathsf{X}_2 | \mathsf{X}_1) P(\mathsf{X}_1) + \Pr(\mathsf{X}_1 | \mathsf{X}_2) P(\mathsf{X}_2) \\ &= \frac{1}{2} \Pr(\mathsf{X}_2 | \mathsf{X}_1) + \frac{1}{2} \Pr(\mathsf{X}_1 | \mathsf{X}_2), \end{aligned}$$

where in the last equality we assumed that X_1 and X_2 are transmitted with equal probability.

[3] For notational simplicity we write $\Pr(\mathsf{X}_2|\mathsf{X}_1)$ as a shorthand notation for $\Pr(\hat{x} = \mathsf{X}_2 | x = \mathsf{X}_1)$.

If $X = \mathsf{X}_1$, Y is Gaussian-distributed with mean $\mu = -\sqrt{\mathsf{E_s}}$ and variance σ^2, $Y \sim \mathcal{N}(-\sqrt{\mathsf{E_s}}, \sigma^2)$, so that

$$\begin{aligned}
\Pr(\hat{x} = \mathsf{X}_2 | x = \mathsf{X}_1\,) &= \Pr(Y > 0 | X = \mathsf{X}_1\,) \\
&= \Pr(Y > 0)\big|_{Y \sim \mathcal{N}(-\sqrt{\mathsf{E_s}}, \sigma^2)} \\
&= \mathsf{Q}\left(\frac{\sqrt{\mathsf{E_s}}}{\sigma}\right) \\
&= \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_s}}{\mathsf{N}_0}}\right) \\
&\boxed{= \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_b}}{\mathsf{N}_0}}\right).}
\end{aligned}$$

Due to symmetry considerations, $\Pr(\hat{x} = \mathsf{X}_1 | x = \mathsf{X}_2\,) = p(\hat{x} = \mathsf{X}_2 | x = \mathsf{X}_1\,)$, so that

$$\underline{P_{\mathsf{s}}^{\mathsf{BPSK}} = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_s}}{\mathsf{N}_0}}\right) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_b}}{\mathsf{N}_0}}\right).} \tag{6.13}$$

Consider now the Euclidean distance between symbols X_1 and X_2, $d_{\mathsf{E}}(\mathsf{X}_1, \mathsf{X}_2) = 2\sqrt{\mathsf{E_s}}$. Using this in (6.13), we can rewrite the symbol error probability as

$$\boxed{P_{\mathsf{s}}^{\mathsf{BPSK}} = \mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_2)}{2\mathsf{N}_0}}\right),}$$

i.e., the symbol error probability depends on the Euclidean distance between constellation points.

The dependence of the symbol error probability on the Euclidean distance can be generalized to other constellations. Consider two complex constellation points $\mathsf{X}_1 \in \mathbb{C}$ and $\mathsf{X}_2 \in \mathbb{C}$ with Euclidean distance $d_{\mathsf{E}}(\mathsf{X}_1, \mathsf{X}_2) = \|\mathsf{X}_1 - \mathsf{X}_2\|$. Assume that $x \in \mathcal{X} = \{\mathsf{X}_1, \mathsf{X}_2, \ldots, \}$ is transmitted over the AWGN channel and we receive $y = x + n$. Now consider a straight line between X_1 and X_2, and the projection of y onto this line, which we denote by $\tilde{y}$. It follows that

$$\tilde{y} = x + \tilde{n}, \tag{6.14}$$

where $\tilde{N} \sim (0, \sigma^2)$ and $\tilde{y}$ contains all the information to make a decision regarding x. If $x = \mathsf{X}_1$ is transmitted, an error occurs if $\tilde{y}$ is closer to X_2 than to X_1. Assume without loss of generality that $\mathsf{X}_1 = (0, 0)$. Then, $\tilde{Y} \sim \mathcal{N}(0, \sigma^2)$ and the probability that the decoder decides for X_2 instead of X_1 is $\Pr(\tilde{Y} > d_{\mathsf{E}}(\mathsf{X}_1, \mathsf{X}_2)/2)$, i.e., $\tilde{y}$ is closer to X_2 than to X_1. Thus,

$$\begin{aligned}
\Pr(\hat{x} = \mathsf{X}_2 | x = \mathsf{X}_1\,) &= \Pr\left(\tilde{Y} > \frac{d_{\mathsf{E}}(\mathsf{X}_1, \mathsf{X}_2)}{2}\right)\Bigg|_{\tilde{Y} \sim \mathcal{N}(0, \sigma^2)} \\
&= \mathsf{Q}\left(\frac{d_{\mathsf{E}}(\mathsf{X}_1, \mathsf{X}_2)}{2\sigma}\right) \\
&= \underline{\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_2)}{2\mathsf{N}_0}}\right).}
\end{aligned} \tag{6.15}$$

Therefore, the probability that the decoder decodes on constellation symbol X_2 if symbol X_1 was transmitted (and ignoring all other constellation symbols), commonly referred to as the **pairwise error probability**, depends on the distance between constellation symbols! Given an average energy per symbol $\mathsf{E_s}$, we will then be interested in placing the constellation points in the real or complex plane such that the Euclidean distance between constellation points is maximized.

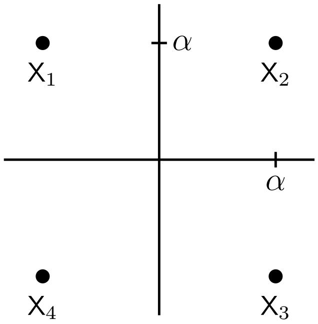


Figure 6.7: 4-QAM constellation. $\alpha = \sqrt{E_s/2}$ such that the energy per symbol is E_s.

6.3.2 Symbol Error Probability of 4-QAM

For 4-QAM (or 4-PSK), we can still derive an exact expression for the symbol error probability. The constellation is $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$, where

$$X_1 = \alpha(-1+j), \quad X_2 = \alpha(1+j), \quad X_3 = \alpha(1-j), \quad \text{and } X_4 = \alpha(-1-j), \tag{6.16}$$

with $\alpha = \sqrt{E_s/2}$ such that the energy per symbol is E_s. The constellation is depicted in Figure 6.7.

The received value is $y = x + n$, with $N \sim \mathcal{CN}(0, 2\sigma^2)$, and the ML decision is

$$\hat{x} = \begin{cases} X_1 & \text{if } y_I < 0 \text{ and } y_Q > 0 \\ X_2 & \text{if } y_I > 0 \text{ and } y_Q > 0 \\ X_3 & \text{if } y_I > 0 \text{ and } y_Q < 0 \\ X_4 & \text{if } y_I < 0 \text{ and } y_Q < 0 \end{cases},$$

where y_I and y_Q are the imaginary and quadrature components of the received value y.

Noting that for 4-QAM the symbol error probability does not depend on the transmitted symbol, we can assume the transmission of X_1. The symbol error probability can then be written as

$$\begin{aligned} P_s^{4QAM} &= \Pr(\hat{x} \neq X_1 | x = X_1) \\ &= \Pr(Y_I > 0 \cup Y_Q < 0 | X_1) \\ &\overset{(a)}{=} \Pr(Y_I > 0|X_1) + \Pr(Y_Q < 0|X_1) - \Pr(Y_I > 0 \cap Y_Q < 0|X_1) \\ &= \Pr(Y_I > 0|X_1) + \Pr(Y_Q < 0|X_1) - \Pr(Y_I > 0|X_1)\Pr(Y_Q < 0|X_1), \end{aligned}$$

where in (a) we used $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$.

Note that if X_1 is transmitted, $Y_I \sim \mathcal{N}(-\alpha, \sigma^2)$ and $Y_Q \sim \mathcal{N}(\alpha, \sigma^2)$. Therefore, we can proceed as

$$\begin{aligned} \Pr(Y_I > 0|X_1) &= \Pr(Y_I > 0)\big|_{Y_I \sim \mathcal{N}(-\alpha,\sigma^2)} \\ &= Q\left(\frac{\alpha}{\sigma}\right) \\ &\overset{(a)}{=} Q\left(\sqrt{\frac{E_s}{N_0}}\right) \\ &= Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \end{aligned}$$

where in (a) we used $\alpha = \sqrt{\mathsf{E_s}/2}$.

By symmetry of the constellation, $\Pr(Y_{\mathsf{Q}} < 0|\mathsf{X}_1) = \Pr(Y_{\mathsf{I}} > 0|\mathsf{X}_1)$, hence the exact symbol error probability of 4-QAM is given by

$$P_{\mathsf{s}}^{4\mathsf{QAM}} = 2\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_b}}{\mathsf{N_0}}}\right) - \left(\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_b}}{\mathsf{N_0}}}\right)\right)^2. \tag{6.17}$$

6.3.3 Upper Bound on the Symbol Error Probability of General M-ary Constellations

For general constellations, the exact symbol error probability P_{s} is hard to compute. We can, however, apply the union bound (see Section 6.2.1) to compute an upper bound on the symbol error probability. We develop P_{s} for an M-ary constellation as

$$\begin{aligned} P_{\mathsf{s}}^{(M)} &= \sum_{i=1}^{M} \Pr(\hat{x} \neq \mathsf{X}_i | x = \mathsf{X}_i) P(\mathsf{X}_i) \\ &= \frac{1}{M}\sum_{i=1}^{M} \Pr(\hat{x} \neq \mathsf{X}_i | x = \mathsf{X}_i) \\ &= \frac{1}{M}\sum_{i=1}^{M} \Pr\left(\bigcup_{j\neq i} \hat{x} = \mathsf{X}_j | x = \mathsf{X}_i\right) \\ &\overset{(a)}{\leq} \frac{1}{M}\sum_{i=1}^{M}\sum_{j\neq i} \Pr(\hat{x} = \mathsf{X}_j | x = \mathsf{X}_i), \end{aligned} \tag{6.18}$$

where in (a)Â we used the union bound (6.8). As we have seen, given two constellation points X_i and X_j, the pairwise error probability $\Pr(\hat{x} = \mathsf{X}_j|x = \mathsf{X}_i)$ depends only on the Euclidean distance between the two constellation points. Thus (see (6.15)),

$$\Pr(\hat{x} = \mathsf{X}_j|x = \mathsf{X}_i) = \mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_i,\mathsf{X}_j)}{2\mathsf{N_0}}}\right).$$

Using this in (6.18), the symbol error probability of an M-ary constellation can be upperbounded as

$$P_{\mathsf{s}}^{(M)} \leq \frac{1}{M}\sum_{i=1}^{M}\sum_{j\neq i} \mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_i,\mathsf{X}_j)}{2\mathsf{N_0}}}\right). \tag{6.19}$$

Example 6.1 (Upper bound on the symbol error probability of 4-QAM) For 4-QAM, with $\mathcal{X}$ given in (6.16) we obtain

$$\begin{aligned} P_{\mathsf{s}}^{4\mathsf{QAM}} &\leq \frac{1}{4}\sum_{i=1}^{4}\sum_{j\neq i} \mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_i,\mathsf{X}_j)}{2\mathsf{N_0}}}\right) \\ &= \sum_{j=2}^{4} Q\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1,\mathsf{X}_j)}{2\mathsf{N_0}}}\right) \\ &= \mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1,\mathsf{X}_2)}{2\mathsf{N_0}}}\right) + Q\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1,\mathsf{X}_3)}{2\mathsf{N_0}}}\right) + Q\left(\sqrt{\frac{d_{\mathsf{E}}^2(\mathsf{X}_1,\mathsf{X}_4)}{2\mathsf{N_0}}}\right). \end{aligned}$$

where in the first equality, due to symmetry in the constellation, we can assume that X_1 was transmitted (i.e., the symbol error probability does not depend on the transmitted symbol).

Now, $d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_2) = \|\alpha(-1+j) - \alpha(1+j)\|^2 = 4\alpha^2 = 2\mathsf{E}_\mathsf{s}$, $d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_4) = d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_2) = 2\mathsf{E}_\mathsf{s}$ due to symmetry, and $d_{\mathsf{E}}^2(\mathsf{X}_1, \mathsf{X}_3) = \|\alpha(-1+j) - \alpha(1-j)\|^2 = 8\alpha^2 = 4\mathsf{E}_\mathsf{s}$. Therefore,

$$\begin{aligned} P_\mathsf{s}^{\mathsf{4QAM}} &\leq 2\mathsf{Q}\left(\sqrt{\frac{\mathsf{E}_\mathsf{s}}{\mathsf{N}_0}}\right) + \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{s}}{\mathsf{N}_0}}\right) \\ &= 2\mathsf{Q}\left(\sqrt{\frac{\mathsf{E}_\mathsf{b}\log_2 M}{\mathsf{N}_0}}\right) + \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}\log M}{\mathsf{N}_0}}\right) \\ &= 2\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right) + \mathsf{Q}\left(\sqrt{\frac{4\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right). \end{aligned} \tag{6.20}$$

Note that (6.20) is larger than the exact value of the symbol error probability of 4-QAM in (6.17). However, comparing (6.20) with (6.17) we see that the dominant term is the same.

6.3.4 Nearest Neighbor Approximation

For large M, even the union bound may be difficult to compute. Furthermore, due to the addition of many terms, the union bound may diverge. A useful alternative is to compute an approximation to the probability of error. Due to the properties of the Q-function (see (6.12)), the dominant terms in (6.19) are the ones with smaller minimum Euclidean distance $d_\mathsf{E}(\mathsf{X}_i, \mathsf{X}_j)$. Therefore, for each X_i we can keep the terms in the inner summation corresponding to the set of constellation symbols $\{\mathsf{X}_j\}$ for which $d_\mathsf{E}(\mathsf{X}_i, \mathsf{X}_j)$ is minimum. Let

$$d_{\mathsf{E,min}}(\mathsf{X}_i) = \min_{\mathsf{X}_j \neq \mathsf{X}_i} d_\mathsf{E}(\mathsf{X}_i, \mathsf{X}_j)$$

and $A_{\mathsf{min}}(\mathsf{X}_i)$ the number of constellation points at distance $d_{\mathsf{E,min}}(\mathsf{X}_i)$ from X_i. The constellation points at distance $d_{\mathsf{E,min}}(\mathsf{X}_i)$ of X_i are usually referred to as the **nearest neighbors** of X_i. Then, $P_\mathsf{s}^{(M)}$ can be approximated as

$$P_\mathsf{s}^{(M)} \approx \frac{1}{M}\sum_{i=1}^{M} A_{\min}(\mathsf{X}_i)\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E,min}}^2(\mathsf{X}_i)}{2\mathsf{N}_0}}\right). \tag{6.21}$$

We can further approximate $P_\mathsf{s}^{(M)}$ by considering only the minimum Euclidean distance. Define

$$\boxed{d_{\mathsf{E,min}} = \min_{\mathsf{X}_i} d_{\mathsf{E,min}}(\mathsf{X}_i).}$$

Then (6.21) can be approximated as

$$
\begin{aligned}
P_{\mathsf{s}}^{(M)} &\approx \frac{1}{M}\sum_{i=1}^{M} A_{\min}(\mathsf{X}_i)\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E},\min}^2(\mathsf{X}_i)}{2\mathsf{N}_0}}\right)\\
&\geq \frac{1}{M}\sum_{i=1}^{M} A_{\min}(\mathsf{X}_i)\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E},\min}^2}{2\mathsf{N}_0}}\right)\\
&= \left(\frac{1}{M}\sum_{i=1}^{M} A_{\min}(\mathsf{X}_i)\right)\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E},\min}^2}{2\mathsf{N}_0}}\right)\\
&\boxed{= \bar{A}_{\min}\mathsf{Q}\left(\sqrt{\frac{d_{\mathsf{E},\min}^2}{2\mathsf{N}_0}}\right),}
\end{aligned}
\tag{6.22}
$$

where

$$
\underline{\bar{A}_{\min} = \frac{1}{M}\sum_{i=1}^{M} A_{\min}(\mathsf{X}_i)}
$$

is the average number of nearest neighbors. We refer to the approximation of the symbol error probability in (6.22) as the **nearest neighbor approximation**. Note that this approximation only requires knowledge of $\bar{A}_{\min}$ and $d_{\mathsf{E},\min}$.

Remark 6.1 The union bound and the nearest neighbor approximation are meaningful (i.e., tight to the actual performance) for high $\mathsf{E_b}/\mathsf{N}_0$.

Example 6.2 (Nearest neighbor approximation for squared M-QAM) For squared M-QAM constellations, there are 4 constellation points with 2 neighbors, $4(\sqrt{M}-2)$ points with 3 neighbors and the remaining points have 4 neighbors, hence $\bar{A}_{\min} = 4 - 4/\sqrt{M}$. Furthermore, $d_{\mathsf{E},\min} = \sqrt{\frac{6\mathsf{E_s}}{M-1}}$. Using this in (6.22) we obtain

$$
P_{\mathsf{s}}^{M\mathsf{QAM}} \approx \left(4-\frac{4}{\sqrt{M}}\right)\mathsf{Q}\left(\sqrt{\frac{3\mathsf{E_s}}{(M-1)\mathsf{N}_0}}\right) = \left(4-\frac{4}{\sqrt{M}}\right)\mathsf{Q}\left(\sqrt{\frac{3\mathsf{E_b}\log M}{(M-1)\mathsf{N}_0}}\right). \tag{6.23}
$$

Example 6.3 (Nearest neighbor approximation for M-PSK) For M-PSK, the constellation is

$$
\mathcal{X} = \{\sqrt{\mathsf{E_s}}\exp(j2\pi i/M) : i = 1,\dots,M\},
$$

so that

$$
d_{\mathsf{E},\min} = \left\|\sqrt{\mathsf{E_s}} - \sqrt{\mathsf{E_s}}\exp(j2\pi/M)\right\| = 2\sqrt{\mathsf{E_s}}\sin(\pi/M).
$$

For $M > 2$, there are two nearest neighbors for any constellation point, i.e., $\bar{A}_{\min} = 2$, while for $M = 1$,

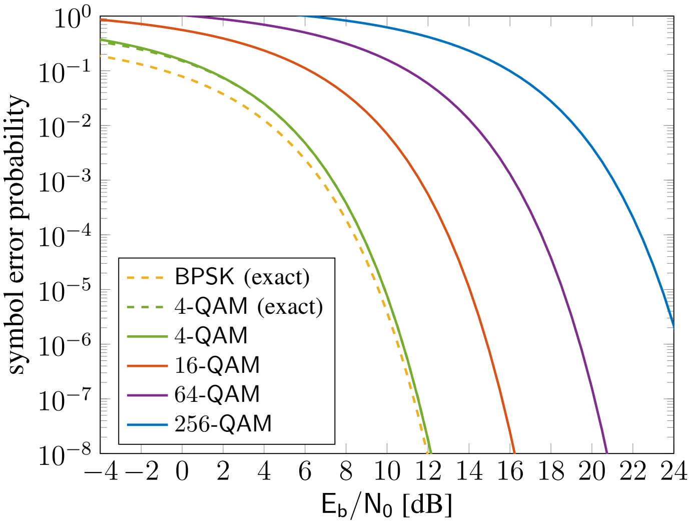


Figure 6.8: Exact symbol error probability of BPSK and 4-QAM and nearest neighbor approximation of the symbol error probability of several QAM modulations.

$\bar{A}_{\mathsf{min}} = 1$. Using this in (6.22) we obtain

$$P_{\mathsf{s}}^{\mathsf{PSK}} \approx \begin{cases} 2\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_s}\sin^2(\pi/M)}{\mathsf{N_0}}}\right) = 2\mathsf{Q}\left(\sqrt{\frac{2\sin^2(\pi/M)\mathsf{E_b}\log_2 M}{\mathsf{N_0}}}\right) & M > 2 \\ \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_s}}{\mathsf{N_0}}}\right) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E_b}}{\mathsf{N_0}}}\right) & M = 2. \end{cases} \tag{6.24}$$

The exact symbol error probability of BPSK (6.13) and 4-QAM (6.17), as well as the nearest neighbor approximation for several QAM constellations according to (6.23) is depicted in Figure 6.8. As can be seen, the nearest neighbor approximation gives a very close approximation for 4-QAM.

From (6.23) and (6.24), and as observed in Figure 6.8, it results that the power efficiency of QAM and PSK diminishes with M, i.e., the $\mathsf{E_b/N_0}$ required to achieve a certain error probability increases with M. On the other hand, the spectral efficiency increases logarithmically with M, $m = \log M$.

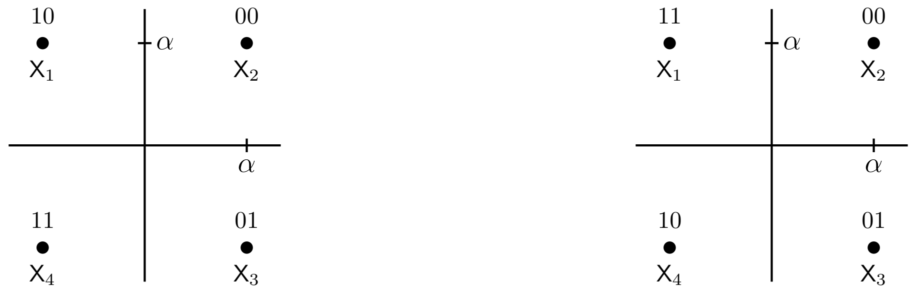
6.4 Bit Error Probability of Linear Modulation Formats

So far we have addressed the computation of the symbol error probability. Sometimes it is more useful to compute the bit error probability. It is important to realize that the bit error probability for a given constellation depends on the so-called **binary labeling**, i.e., **how tuples of $m = \log M$ bits are mapped to the constellation symbols**. Going back to the communication system in Figure 6.3, if an M-ary modulation is used for transmission, then the data sequence $\boldsymbol{u}$ is parsed into blocks of of $m = \log M$ bits, which are mapped to the constellation symbols according to a given binary labeling. Let

$$\mathsf{L}(x) = (b_1(x), \ldots, b_m(x))$$

be the m-bit labeling associated to constellation symbol $x \in \mathcal{X}$. For instance, for the example of Figure 6.1, Gray labeling is used and $\mathsf{L}(+1) = (1, 1, 0)$. For later use, we also define the mapping

$$\mathsf{L}^{-1} : \{0, 1\}^m \to \mathcal{X},$$



(a) 4-QAM, Gray labeling.

(b) 4-QAM, lexicographic labeling.

Figure 6.9: 4-QAM constellation with two different binary labelings.

where $\mathsf{L}^{-1}(b_1,\ldots,b_m)$ is the constellation symbol corresponding to the binary tuple $(b_1,\ldots,b_m)$. For instance, for the example of Figure 6.1, $\mathsf{L}^{-1}(011)=-3$.

Obviously, the way tuples of m bits are mapped to constellation symbols is not unique. Also, not surprisingly, different labelings lead to different bit error probabilities!

The bit error probability P_{b} of an M-ary constellation is given by

$$\boxed{P_{\mathsf{b}}=\frac{1}{m}\sum_{i=1}^{m}\Pr(i\text{th bit in error}).}$$

For ease of notation, we define $p_i\triangleq\Pr(i\text{th bit in error})$ and by e_{b_i} the event that the i-th bit is decoded in error.

In the following, we derive the exact bit error probability of 4-QAM assuming two different bit mappings and then we consider the general case of M-ary constellations.

6.4.1 Bit Error Probability of 4-QAM

Consider 4-QAM ($m=2$, i.e., $\mathsf{L}(x)=(b_1(x),b_2(x))$) and two possible bit labelings, Gray labeling and lexicographic labeling, shown in Figure 6.9. The lexicographic labeling corresponds to the lexicographic binary representation of 0, 1, 2, and 3, numbering the symbols in clockwise order. We will see in the following that the two labelings lead to different P_{b}.

Bit Error Probability of 4-QAM with Gray Labeling

We first compute p_1 and p_2,

$$\begin{aligned}
p_1&=\Pr(\mathsf{e}_{b_1})\\
&=\sum_{x\in\mathcal{X}}\Pr(\mathsf{e}_{b_1}|x)P(x)\\
&\overset{(a)}{=}\frac{1}{4}\sum_{x\in\mathcal{X}}\Pr(\mathsf{e}_{b_1}|x)\\
&=\frac{1}{4}(\Pr(\mathsf{e}_{b_1}|\mathsf{X}_1)+\Pr(\mathsf{e}_{b_1}|\mathsf{X}_2)+\Pr(\mathsf{e}_{b_1}|\mathsf{X}_3)+\Pr(\mathsf{e}_{b_1}|\mathsf{X}_4))\\
&=\frac{1}{4}(\Pr(b_1(\hat{x})=0|\mathsf{X}_1)+\Pr(b_1(\hat{x})=1|\mathsf{X}_2)+\Pr(b_1(\hat{x})=1|\mathsf{X}_3)+\Pr(b_1(\hat{x})=0|\mathsf{X}_4)),\qquad(6.25)
\end{aligned}$$

where $\Pr(\mathsf{e}_{b_1}|x)$ is the probability that the first bit is decoded in error if x was transmitted, $\hat{x}$ is the decoded symbol, and in (a) we have exploited the fact that all symbols are transmitted with equal probability, i.e., $P_X(\mathsf{X}_i) = 1/M = 1/4$ for all i.

Now,

$$\begin{aligned}\Pr(b_1(\hat{x}) = 0|\mathsf{X}_1) &= p(Y_\mathsf{I} > 0|\mathsf{X}_1)\\ &= \Pr(Y_\mathsf{I} > 0)\big|_{Y_\mathsf{I} \sim \mathcal{N}(-\alpha,\sigma^2)}\\ &= \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right),\end{aligned}$$

and $\Pr(b_1(\hat{x}) = 0|\mathsf{X}_4) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)$.

Similarly,

$$\begin{aligned}\Pr(b_1(\hat{x}) = 1|\mathsf{X}_2) &= p(Y_\mathsf{I} < 0|\mathsf{X}_2)\\ &= \Pr(Y_\mathsf{I} < 0)\big|_{Y_\mathsf{I} \sim \mathcal{N}(\alpha,\sigma^2)}\\ &= \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right),\end{aligned}$$

and $\Pr(b_1(\hat{x}) = 1|\mathsf{X}_3) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)$.

Using this in (6.25), we obtain

$$p_1 = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right).$$

It is also easy to see that, due to symmetry, $p_2 = p_1$. Therefore, the bit error probability of 4-QAM with Gray labeling is

$$\boxed{P_\mathsf{b}^{\mathsf{4QAM-Gray}} = \frac{1}{2}(p_1 + p_2) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right).} \tag{6.26}$$

Notice that the average bit error probability of 4-QAM and Gray labeling is **identical** to the bit error probability of BPSK (see Section 6.3.1).[4] The reason is that 4-QAM with Gray labeling can be seen as two independent BPSK constellations, for the in-phase and quadrature components.

Bit Error Probability of 4-QAM with Lexicographic Labeling

The bit error probability can be calculated as we did for the Gray labeling. However, note that in this case, due to the lack of symmetry, $p_1 \neq p_2$. After some basic calculations, we obtain[5]

$$p_1 = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right) \tag{6.27}$$

$$p_2 = 2\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)\left(1 - \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)\right). \tag{6.28}$$

Thus,

$$P_\mathsf{b}^{\mathsf{4QAM-Lex}} = \frac{1}{2}(p_1 + p_2) = \frac{3}{2}\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right) - \left(\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)\right)^2. \tag{6.29}$$

[4] For BPSK the symbol error probability and bit error probability are obviously identical, since each symbol carries a single bit.

[5] The computation of P_b for the lexicographical labeling is left as an exercise.

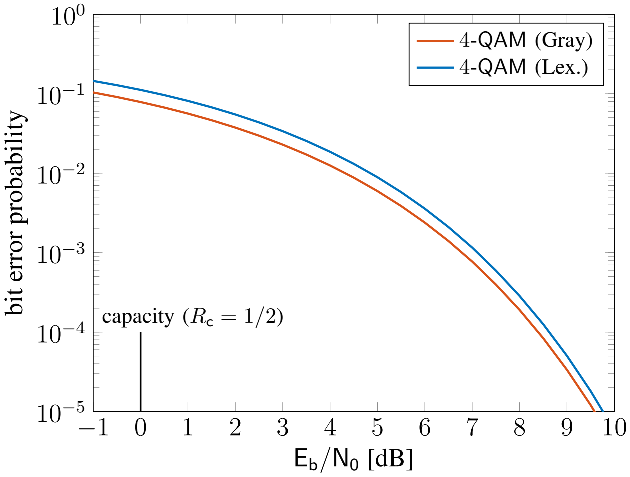


Figure 6.10: Bit error probability of uncoded 4-QAM with Gray and lexicographic labeling as a function of $\mathsf{E_b/N_0}$ and comparison with the channel capacity.

Comparing (6.29) and (6.26) we note that the bit error probability of 4-QAM with Gray labeling is lower. Also, note that for the lexicographic labeling the first bit suffers from fewer errors. While for uncoded systems there is no advantage in using such unequal error protection, sophisticated labelings play an important role in coded modulation schemes.

In Figure 6.10, we plot the bit error probability of 4-QAM with both Gray labeling and lexicographic labeling as a function of $\mathsf{E_b/N_0}$. The figure shows that different (non-equivalent) bit labelings yield different bit error probabilities. Note that Gray labeling yields better performance. This is in general the case for linear modulations and uncoded transmission. For comparison purposes, we also plot the channel capacity in the figure. Note that to achieve a probability of error of 10^{-5} we require around 9.5 dB, i.e., uncoded BPSK and 4-QAM with Gray labeling perform around 7.8 dB away from capacity to achieve $P_{\mathsf{b}} = 10^{-5}$.

6.4.2 Bit Error Probability of M-ary Constellations: Nearest Neighbor Approximation

For general M-ary constellations and arbitrary labelings, the computation of P_{b} is cumbersome. However, as for the symbol error probability, we can approximate P_{b} by using the nearest neighbor approximation. Let us focus on bit b_i, and compute p_i, the probability of making an error on the i-th bit,

$$\begin{aligned} p_i &= \Pr(\mathsf{e}_{b_i}) \\ &= \sum_{x\in\mathcal{X}} \Pr(\mathsf{e}_{b_i}|x)P(x) \\ &= \frac{1}{M}\sum_{x\in\mathcal{X}} \Pr(\mathsf{e}_{b_i}|x). \end{aligned} \tag{6.30}$$

Note that

$$\Pr(\mathsf{e}_{b_i}|x) = \Pr\left(\bigcup_{\substack{\hat{x}\neq x\\ b_i(\hat{x})\neq b_i(x)}} \hat{X} = \hat{x}|X = x\right). \tag{6.31}$$

Thus, computing $\Pr(\mathsf{e}_{b_i}|x)$ exactly is in general a difficult task, since it requires to compute the probability of the union of many events. Instead of doing so, we can approximate (6.31) by considering only the neighboring constellation points of x whose binary labels differ from the binary label of x, $\mathsf{L}(x)$, in the i-th position. For instance, for the Gray labeling in Figure 6.9(a) we can approximate $\Pr(\mathsf{e}_{b_1}|\mathsf{X}_2)$ by considering the only neighboring constellation point for which the first bit differs from $b_1(\mathsf{X}_2) = 0$, i.e., X_1 (in this case $b_1(\mathsf{X}_1) = 1 \neq b_1(\mathsf{X}_2)$). Note that it is more likely that an error will be made by decoding to a neighboring symbol of x rather than to a more distant symbol! We formalize this in the following.

As in Section 6.3.4, for two constellation symbols $x \in \mathcal{X}$ and $\tilde{x} \in \mathcal{X}$, let

$$d_{\mathsf{E,min}}(x) = \min_{\tilde{x} \neq x} d_{\mathsf{E}}(x, \tilde{x}).$$

Also, denote by $A^{\mathsf{b}}_{\mathsf{min}}(x, i)$ the number of nearest neighbors of constellation symbol $x \in \mathcal{X}$ that differ in the i-th bit position, i.e.,

$$A^{\mathsf{b}}_{\mathsf{min}}(x, i) = |\{\tilde{x} : d_{\mathsf{E}}(x, \tilde{x}) = d_{\mathsf{E,min}}(x), b_i(x) \neq b_i(\tilde{x})\}|.$$

Using $d_{\mathsf{E,min}}(x)$ and $A^{\mathsf{b}}_{\mathsf{min}}(x, i)$, the conditional probability of making an error on the i-th bit if constellation symbol $x \in X$ was transmitted can be approximated as

$$\begin{aligned}\Pr(\mathsf{e}_{b_i}|x) &\approx A^{\mathsf{b}}_{\mathsf{min}}(x, i)\mathsf{Q}\left(\frac{d_{\mathsf{E,min}}(x)}{2\sigma}\right)\\ &= A^{\mathsf{b}}_{\mathsf{min}}(x, i)\mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}(x)}{2\mathsf{N}_0}}\right). \qquad (6.32)\end{aligned}$$

Substituting (6.32) in (6.30) we obtain

$$p_i \approx \frac{1}{M} \sum_{x \in \mathcal{X}} A^{\mathsf{b}}_{\mathsf{min}}(x, i)\mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}(x)}{2\mathsf{N}_0}}\right). \qquad (6.33)$$

We can further approximate (6.33) by considering

$$d_{\mathsf{E,min}} = \min_{x} d_{\mathsf{E,min}}(x),$$

thus obtaining

$$\begin{aligned}p_i &\approx \mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}}{2\mathsf{N}_0}}\right)\frac{1}{M}\sum_{x \in \mathcal{X}} A^{\mathsf{b}}_{\mathsf{min}}(x, i)\\ &= \bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_i)\mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}}{2\mathsf{N}_0}}\right), \qquad (6.34)\end{aligned}$$

where

$$\bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_i) = \frac{1}{M}\sum_{x \in \mathcal{X}} A^{\mathsf{b}}_{\mathsf{min}}(x, i). \qquad (6.35)$$

Example 6.4 Consider 4-QAM and lexicographic labeling. For 4-QAM, $d_{\mathsf{E,min}} = \sqrt{2\mathsf{E}_\mathsf{s}}$. We compute the approximation of p_1 and p_2 according to (6.34). Notice that $A^{\mathsf{b}}_{\mathsf{min}}(x, 1) = 1$ for all x while $A^{\mathsf{b}}_{\mathsf{min}}(x, 2) = 2$ for all x, hence $\bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_1) = 1$ and $\bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_2) = 2$. It follows

$$p_1 \approx \bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_1)\mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}}{2\mathsf{N}_0}}\right) = \mathsf{Q}\left(\sqrt{\frac{\mathsf{E}_\mathsf{s}}{\mathsf{N}_0}}\right) = \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right),$$

and

$$p_2 \approx \bar{A}^{\mathsf{b}}_{\mathsf{min}}(b_2)\mathsf{Q}\left(\sqrt{\frac{d^2_{\mathsf{E,min}}}{2\mathsf{N}_0}}\right) = 2\mathsf{Q}\left(\sqrt{\frac{\mathsf{E}_\mathsf{s}}{\mathsf{N}_0}}\right) = 2\mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right).$$

As can be seen, the approximation of p_1 is exact, while the approximation of p_2 slightly overestimates its value (cf. (6.27) and (6.28)).

Chapter 7

Basics of Error Correcting Coding

We have seen in the previous chapter that uncoded transmission performs very far away from capacity, i.e., uncoded transmission requires a much larger $\mathsf{E_b/N_0}$ to achieve reliable communication than the fundamental limit established by Shannon. Fortunately, Shannon also provided the way to achieve capacity: he proved that capacity can be achieved by the use of **coding**. The principle of coding is very simple: **introducing redundancy** to the transmitted sequence in a controlled manner such that it can be exploited by the receiver to correct the errors introduced by the channel. Since the purpose of the code is to correct errors, (channel) coding is usually referred to as **error correcting coding**.

While Shannon proved the **existence** of capacity-achieving codes, he did not provide any insight on how to construct **practical codes**. His proof was based on a random-coding argument: roughly speaking, pick a code randomly and, with high probability, it will achieve capacity. Unfortunately, random codes are not practical, as both encoding and decoding require computational complexity and storage that grow exponentially with the block length (and Shannon showed that to achieve capacity, large block lengths —in truth infinitely large—are required!). Therefore, since the late 1940s, finding practical codes that are able to perform close to capacity yet with reasonable decoding complexity has been the holy grail of coding theory. For decades, researchers developed powerful codes with a rich algebraic structure, among them the celebrated Bose-Chaudhuri-Hocquenghem (BCH) codes and Reed-Solomon codes, which unfortunately still performed far away from the channel capacity with feasible decoding algorithms. The breakthrough came in the early 1990s, when two French researchers, Claude Berrou and Alain Glavieux, introduced turbo codes, the first class of codes that was shown to approach capacity with reasonable decoding complexity, and with the rediscovery of low-density parity-check (LDPC) codes (originally introduced in 1967 by Robert Gallager) by David MacKay. The advent of turbo codes and the rediscovery of LDPC codes constituted a cornerstone in coding theory and gave birth to what nowadays is referred to as the field of **modern coding theory**, which is intimately related to the iterative decoding of these codes. LDPC codes and turbo codes have now been adopted in a plethora of communication standards.

The application of coding goes far beyond the classical communication and storage problem. Coding is a very powerful, fundamental tool that plays a key role in a myriad of other applications and research fields, such as distributed storage and caching [1, 2], uncoordinated multiple-access [3], to speed-up distributed computing tasks [4, 5], quantum key distribution [6], and post-quantum cryptography [7].

In this chapter, we introduce the basic concepts underlying coding. Our focus is on binary codes, since due to their lower decoding complexity with respect to nonbinary codes they are by far the most popular and widely used (except, perhaps, Reed-Solomon codes). However, most of the concepts introduced in this chapter extend to nonbinary codes in a straightforward manner. In the subsequent Chapters 8 and 9, we will then discuss two of the fundamental coding schemes, linear block codes and convolutional codes, which are at the basis of all modern coding schemes. Finally, in Chapters 10 and 11 we will briefly introduce two of the most important classes of modern codes, turbo codes and LDPC codes, which are adopted in many current communication standards.

7.1 Error Correcting Code and Encoder

Definition 7.1 (Error correcting code) A binary block code of code length n and dimension k, denoted by $\mathcal{C}(n,k)$ or simply by the pairÂ (n,k), is a collection of 2^k binary tuples of length n bits,

$$\mathcal{C}(n,k) = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \dots, \boldsymbol{c}_{2^k} : \boldsymbol{c}_i \in \{0,1\}^n\}.$$

The n-tuples $\boldsymbol{c}_1, \dots, \boldsymbol{c}_{2^k}$ are called **codewords** and k is also referred to as the **information block length** of the code.

Definition 7.2 (Encoder) An **encoder** $\mathcal{E}$ is a set of 2^k pairs $(\boldsymbol{u}, \boldsymbol{c})$, where $\boldsymbol{u}$ is the information word of length k bits and $\boldsymbol{c}$ is the codeword of length n bits. In other words, the encoder refers to the one-to-one correspondence between information words and codewords, and consists of

(i) 2^k codewords belonging to a set $\mathcal{C} \subset \{0,1\}^n$,

(ii) A mapping function from $\{0,1\}^k$ to $\mathcal{C}$ that maps each sequence of k **information bits** $\boldsymbol{u} = (u_1, \dots, u_k) \in \{0,1\}^k$ into a **codeword** of n **coded bits** $\boldsymbol{c} = (c_1, \dots, c_n) \in \mathcal{C}$.

An important parameter is the **code rate** R_{c}, or simply rate, defined as

$$R_{\mathsf{c}} \triangleq \frac{k}{n} < 1.$$

The rate is a measure of the redundancy of the code. Since $R_{\mathsf{c}} < 1$, then $n > k$, i.e., the information words are encoded into longer sequences, thereby introducing redundancy.

Example 7.1 ((3, 1) Repetition Code) Consider the 3-repetition code with parameters $(n,k) = (3,1)$ consisting of $2^k = 2$ codewords of length 3 bits,

$$\mathcal{C}_{\mathsf{rep}}(n=3, k=1) = \{(0,0,0), (1,1,1)\}\,.$$

One may consider an *encoder* that assigns to the information bit $u = 0$ the codeword $(0,0,0)$ and to the information bit $u = 1$ the codeword $(1,1,1)$, i.e.,

$$\begin{aligned} u = 0 \quad &\rightarrow \quad \boldsymbol{c}_1 = (0,0,0) \\ u = 1 \quad &\rightarrow \quad \boldsymbol{c}_2 = (1,1,1). \end{aligned}$$

The assignment of codewords to information words (in this case to a single bit) is usually referred to as *encoding*. The code rate is $R_{\mathsf{c}} = 1/3$.

Observe that the number of codewords of an (n,k) code is typically much smaller than 2^n. For example, a rate $R_{\mathsf{c}} = 1/2$ code with $k = 1024$ has 2^{1024} codewords, while the number of possible bit sequences of length $n = k/R_\mathsf{c} = 2048$ is 2^{2048}. Hence, only one out of 2^{1024} sequences in $\{0,1\}^n$ is used. This is the basic principle of coding: Since the number of codewords (2^k n-bit sequences) is small compared to the total number of n-bit sequences, we can place them far apart from each other in the n-dimensional space. This basic principle allows to correct the errors introduced by the channel.

The code rate is directly related to the spectral efficiency R (in bits per symbol). For BPSK, the encoder maps sequences of k bits into sequences of n bits, which are then modulated onto sequences of modulation symbols, each one carrying a single bit. Therefore, $R = k/n = R_\mathsf{c}$ bits per symbol, or in words, for binary transmission the spectral efficiency is equal to the code rate. We can also express $\mathsf{E_b}/\mathsf{N_0}$ and $\mathsf{E_s}$ as a function of the code rate,

$$\frac{\mathsf{E_b}}{\mathsf{N_0}} = \frac{\mathsf{E_s}}{\mathsf{N_0}}\frac{1}{R_\mathsf{c}}$$

$$\mathsf{E_s} = \mathsf{E_b} R_\mathsf{c}.$$

As the code rate R_c decreases (e.g., larger n for fixed k), the spectral efficiency decreases, and, for a fixed power efficiency, the energy per symbol $\mathsf{E_s}$ also decreases. Hence, it is desirable to construct codes with **high rate**. This is intuitive: We would like to construct powerful codes (able to correct as many errors as possible) introducing little redundancy to the information sequence.

7.2 Minimum Hamming Distance

We now define one of the most important parameters of error correcting codes, the **minimum Hamming distance**. Intuitively, good codes will have codewords that are well separated in the n-dimensional space $\{0,1\}^n$ (we have to be able to distinguish them). The separation is related to the **Hamming distance** between codewords. We will need first the definition of Hamming weight and Hamming distance.

Definition 7.3 (Hamming weight) For a binary vector $\boldsymbol{c} = (c_1, \ldots, c_n)$ of length n, the Hamming weight, denoted by $w_\mathsf{H}(\boldsymbol{c})$, is the number of entries in which $c_i = 1$, i.e.,

$$w_\mathsf{H}(\boldsymbol{c}) = |\{c_i = 1\}|.$$

Definition 7.4 (Hamming distance) For any two binary vectors $\boldsymbol{c}$ and $\tilde{\boldsymbol{c}}$ of length n, the Hamming distance, denoted by $d_\mathsf{H}(\boldsymbol{c}, \tilde{\boldsymbol{c}})$, is the number of entries in which $\boldsymbol{c}$ and $\tilde{\boldsymbol{c}}$ differ.

It is easy to see that

$$d_\mathsf{H}(\boldsymbol{c}, \tilde{\boldsymbol{c}}) = w_\mathsf{H}(\boldsymbol{c} + \tilde{\boldsymbol{c}}), \tag{7.1}$$

where + denotes binary addition.

Example 7.2 The binary vector $(0,1,1)$ has weight $w_\mathsf{H}(011) = 2$. The two binary vectors $(1,0,0)$ and $(0,1,0)$ differ in two positions, i.e., $d_\mathsf{H}(100, 010) = 2$. Furthermore, $d_\mathsf{H}(100, 010) = w_\mathsf{H}((100) + (010)) = w_\mathsf{H}(110) = 2$.

Definition 7.5 (Minimum Hamming distance) The minimum Hamming distance of a code $\mathcal{C}$, denoted by $d_{\mathsf{min}}(\mathcal{C})$, is defined as

$$d_{\mathsf{min}}(\mathcal{C}) \triangleq \min_{\substack{\boldsymbol{c},\tilde{\boldsymbol{c}}\in\mathcal{C}\\ \boldsymbol{c}\neq\tilde{\boldsymbol{c}}}} d_{\mathsf{H}}(\boldsymbol{c},\tilde{\boldsymbol{c}}).$$

As we will see in the next chapter, the minimum Hamming distance is a very relevant parameter related to the **error correction (and detection) capabilities** of the code. Since the notion of minimum Hamming distance is so important in coding, an (n,k) code with minimum Hamming distance d_{min} is sometimes denoted as an (n,k,d_{min}) code. The field of coding theory involves the analysis and design of codes that have large minimum Hamming distance while at the same time allow for low-complexity encoding and decoding.

Example 7.3 The minimum Hamming distance of the 3-repetition code of Example 7.1 is $d_{\mathsf{min}}(\mathcal{C}_{\mathsf{rep}}) = 3$.

7.3 Linear Block Codes

Almost all practical codes that are in use today are linear codes. The reason is that linear codes are sufficient to achieve capacity and linearity allows for efficient encoding and decoding. All codes considered in this course are linear codes.

A very important subclass of linear codes are linear block codes, defined in the following.

Definition 7.6 (Linear block code) A binary block code $\mathcal{C}(n,k)$ is **linear** if and only if its 2^k codewords $\mathbf{c}_1,\dots,\mathbf{c}_k$ form a k-dimensional subspace of the n-dimensional vector space $\{0,1\}^n$.

For a linear block code, the *all-zero* codeword $(0,0,\dots,0)$ is always a codeword, i.e., $(0,0,\dots,0)\in\mathcal{C}$, and the (binary) addition of two codewords is also a codeword, i.e., for $\boldsymbol{c}\in\mathcal{C}$ and $\tilde{\boldsymbol{c}}\in\mathcal{C}$, then $\boldsymbol{c}+\tilde{\boldsymbol{c}}\in\mathcal{C}$. In other words, the set of codewords is closed under component-wise modulo-2 addition. Furthermore, the minimum Hamming distance of a linear block code is equal to the minimum codeword weight, i.e.,

$$d_{\mathsf{min}}(\mathcal{C}) = \min_{\substack{\boldsymbol{c}\in\mathcal{C}\\ \boldsymbol{c}\neq\boldsymbol{0}}} w_{\mathsf{H}}(\boldsymbol{c}).$$

Proving this equality is straightforward,

$$\begin{aligned} d_{\mathsf{min}}(\mathcal{C}) &= \min_{\substack{\boldsymbol{c},\tilde{\boldsymbol{c}}\in\mathcal{C}\\ \boldsymbol{c}\neq\tilde{\boldsymbol{c}}}} d_{\mathsf{H}}(\boldsymbol{c},\tilde{\boldsymbol{c}})\\ &= \min_{\substack{\boldsymbol{c},\tilde{\boldsymbol{c}}\in\mathcal{C}\\ \boldsymbol{c}\neq\tilde{\boldsymbol{c}}}} w_{\mathsf{H}}(\boldsymbol{c}+\tilde{\boldsymbol{c}})\\ &= \min_{\substack{\boldsymbol{c}'\in\mathcal{C}\\ \boldsymbol{c}'\neq\boldsymbol{0}}} w_{\mathsf{H}}(\boldsymbol{c}'), \end{aligned}$$

where in the last equality we used the fact that the code is linear, i.e., $\boldsymbol{c}'=\boldsymbol{c}+\tilde{\boldsymbol{c}}\in\mathcal{C}$.

It is easy to see that the 3-repetition code of Example 7.1 is a linear code.

A simple upper bound on the minimum Hamming distance of linear codes is given by the **Singleton bound**.

Proposition 7.1 (Singleton bound) *The minimum Hamming distance of a linear code $\mathcal{C}(n,k)$ satisfies the inequality*

$$d_{\mathsf{min}} \le n-k+1.$$

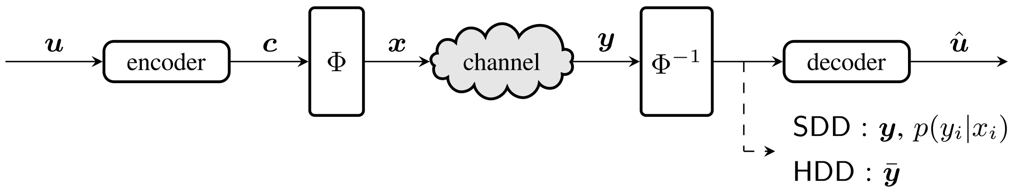


Figure 7.1: Block diagram of a communication system. For soft-decision decoding (SDD), the decoder estimates the transmitted codeword based on $\boldsymbol{y}$ or the transition probabilities $p(y_i|x_i)$, while for hard-decision decoding (HDD) the decoder estimates the transmitted codeword based on the sequence $\bar{\boldsymbol{y}}$.

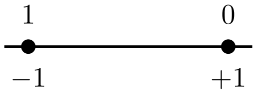


Figure 7.2: Binary phase shift keying constellation with $\mathsf{E_s} = 1$ and mapping $0 \to +1$ and $1 \to -1$.

7.4 Optimum Decoding of Linear Block Codes

We now derive the optimum decoder for block codes. Let us consider again the communication system in Figure 5.1, which is reproduced in Figure 7.1 for convenience. The information word $\boldsymbol{u} = (u_1, \ldots, u_k)$ of length k bits is encoded by a linear block encoder into codeword $\boldsymbol{c} = (c_1, \ldots, c_n)$ of length n bits. We assume binary transmission using BPSK modulation with $\mathsf{E_s} = 1$, i.e., $\mathsf{X}_1 = -1$ and $\mathsf{X}_2 = +1$ (see Figure 6.6). The codeword $\boldsymbol{c}$ is then modulated onto the sequence of constellation symbols $\boldsymbol{x} = (x_1, \ldots, x_n)$ by the modulator Φ, which modulates every code bit c_i of the codeword as $x_i = (-1)^{c_i}$, i.e., without loss of generality, we consider the mapping $0 \to +1$ and $1 \to -1$, as shown in Figure 7.2. For instance, the codeword $\boldsymbol{c} = (0, 1, 1, 1, 0, 0, \ldots)$ is modulated onto $\boldsymbol{x} = (+1, -1, -1, -1, +1, +1, \ldots)$. We assume transmission over a memoryless AWGN channel with zero mean and noise variance σ^2. At the output of the channel, the noisy vector is

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n},$$

where $\boldsymbol{n} = (n_1, \ldots, n_n)$ is the noise vector, with $N_i \sim \mathcal{N}(0, \sigma^2)$.

As for the uncoded case (see Section 6.1), the role of the decoder is to **estimate** the transmitted codeword $\boldsymbol{c}$ (from which we can obtain $\boldsymbol{u}$) optimally based on the noisy oberservation $\boldsymbol{y}$. If all codewords are equiprobable (this is the usual case), as discussed in Section 6.1, the optimum decoding rule is the ML rule,

$$\boxed{\hat{\boldsymbol{x}}_{\mathsf{ML}} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x}),} \tag{7.2}$$

Since the mapping between codewords $\boldsymbol{c}$ and modulated sequences $\boldsymbol{x}$ is invertible, the ML decoding rule in (7.2) is equivalent to

$$\boxed{\hat{\boldsymbol{c}}_{\mathsf{ML}} = \arg\max_{\boldsymbol{c} \in \mathcal{C}} p(\boldsymbol{y}|\boldsymbol{c}).} \tag{7.3}$$

For ease of notation, in the following we will drop the subindex ML.

We need to distinguish between **soft-decision decoding** and **hard-decision decoding**, depending on whether the decoder exploits all the information available at the output of the channel or not. We refer to soft-decision decoding when the decoder estimates $\boldsymbol{c}$ based on the full observation $\boldsymbol{y}$, i.e., when the sequence of (real) values $\boldsymbol{y}$ is fed to the decoder or, equivalently, the decoder is fed with the transition probabilities $p(y_i|x_i)$. Alternatively, we may consider a simpler decoder where the demodulator takes **hard decisions** at the channel output and the sequence of hard-detected symbols, denoted by $\bar{\boldsymbol{y}}$, is fed to the decoder. This is discussed in more detail in the next subsection. The communication system with soft-decision decoding and hard-decision decoding is depicted in Figure 7.1.

7.4.1 AWGN channel with Hard Decisions

Assume that each received value y_i is quantized to two levels, zero or one, according to (see Figure 7.2)

$$\bar{y}_i = \begin{cases} 1 & y_i < 0 \\ 0 & y_i \geq 0. \end{cases}$$

We then say that the demodulator takes hard decisions on the received values.

It is easy to see that, if hard decisions are performed at the output of the channel, the *equivalent* channel between the encoder and the decoder, i.e., the channel with input $\boldsymbol{c}$ and output $\bar{\boldsymbol{y}}$ (see Figure 7.1) is a discrete memoryless channel with bits at the input and bits at the output.

We need to consider the transition probabilities

$$\begin{aligned} &\Pr(\bar{y}_i = 0|c_i = 1), \qquad &\Pr(\bar{y}_i = 1|c_i = 1), \\ &\Pr(\bar{y}_i = 1|c_i = 0), \qquad &\Pr(\bar{y}_i = 0|c_i = 0). \end{aligned}$$

Let us compute first $\Pr(\bar{y}_i = 0|c_i = 1)$. We proceed as

$$\begin{aligned} \Pr(\bar{y}_i = 0|c_i = 1) &= \Pr(\bar{y}_i = 0|x_i = -1) \\ &= \Pr(y_i > 0|x_i = -1) \\ &= \mathsf{Q}\left(\sqrt{\frac{2\mathsf{E}_\mathsf{s}}{\mathsf{N}_0}}\right) \\ &\boxed{= \mathsf{Q}\left(\sqrt{\frac{2R_\mathsf{c}\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right),} \end{aligned}$$

and thus

$$\begin{aligned} \Pr(\bar{y}_i = 1|c_i = 1) &= 1 - \Pr(\bar{y}_i = 0|c_i = 1) \\ &\boxed{= 1 - \mathsf{Q}\left(\sqrt{\frac{2R_\mathsf{c}\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right).} \end{aligned}$$

Due to symmetry,

$$\boxed{\Pr(\bar{y}_1 = 1|c_i = 0) = \Pr(\bar{y}_1 = 0|c_i = 1) = \mathsf{Q}\left(\sqrt{\frac{2R_\mathsf{c}\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)}$$

and

$$\boxed{\Pr(\bar{y}_1 = 0|c_i = 0) = 1 - \Pr(\bar{y}_1 = 1|c_i = 0) = 1 - \mathsf{Q}\left(\sqrt{\frac{2R_\mathsf{c}\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right).}$$

Defining $\varepsilon \triangleq \mathsf{Q}\left(\sqrt{\frac{2R_\mathsf{c}\mathsf{E}_\mathsf{b}}{\mathsf{N}_0}}\right)$, the channel between the encoder output, $\boldsymbol{c}$, and the decoder input, $\bar{\boldsymbol{y}}$, is described by the transition probabilities

$$\begin{aligned}
\Pr(\bar{y}_i = 0|c_i = 1) &= \varepsilon \\
\Pr(\bar{y}_i = 1|c_i = 1) &= 1-\varepsilon \\
\Pr(\bar{y}_i = 1|c_i = 0) &= \varepsilon \\
\Pr(\bar{y}_i = 0|c_i = 0) &= 1-\varepsilon.
\end{aligned}$$

These transition probabilities correspond to those of a BSC, which we introduced in Chapter 4 and is depicted in Figure 4.4. In other words, taking hard decisions at the channel output transforms the AWGN channel (for BPSK transmission) into a BSC with crossover probability

$$\varepsilon = \mathsf{Q}\left(\sqrt{\frac{2R_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right). \tag{7.4}$$

7.4.2 Hard-Decision Decoding

We now derive the optimum decoder for hard-decision decoding, where the decoder estimates the transmitted codeword based on the binary sequence of quantized values $\bar{\boldsymbol{y}} = (\bar{y}_1, \ldots, \bar{y}_n)$. We start again from the ML decoding rule (7.3). Assuming a memoryless channel, i.e., $p(\bar{\boldsymbol{y}}|\boldsymbol{c}) = \prod_{i=1}^n p(\bar{y}_i|c_i)$, we can proceed as

$$\begin{aligned}
\hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} p(\bar{\boldsymbol{y}}|\boldsymbol{c}) \\
&= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \prod_{i=1}^{n} p(\bar{y}_i|c_i).
\end{aligned} \tag{7.5}$$

As we have seen, the channel can be modeled as a BSC with crossover probability ε given in (7.4), i.e.,

$$p(\bar{y}_i|c_i) = \begin{cases} \varepsilon & \bar{y}_i \neq c_i \\ 1-\varepsilon & \bar{y}_i = c_i \end{cases}.$$

Using this in (7.5),

$$\begin{aligned}
\hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \varepsilon^{d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})}(1-\varepsilon)^{n-d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})} \\
&\overset{(a)}{=} \arg\max_{\boldsymbol{c}\in\mathcal{C}} \log\left(\varepsilon^{d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})}(1-\varepsilon)^{n-d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})}\right) \\
&= \arg\max_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})\log\varepsilon + (n - d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}}))\log(1-\varepsilon) \\
&= \arg\max_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})\log\left(\frac{\varepsilon}{1-\varepsilon}\right) + n\log(1-\varepsilon) \\
&\overset{(b)}{=} \arg\max_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})\log\left(\frac{\varepsilon}{1-\varepsilon}\right) \\
&\overset{(c)}{=} \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}}),
\end{aligned} \tag{7.6}$$

where (a) holds because $\log(\cdot)$ is a monotonically increasing function, in (b) we have exploited the fact that addition of a constant does not change the maximization, and in (c), without loss of generality, we assumed that $\varepsilon < 0.5$.

We see from (7.6) that for hard-decision decoding the ML decoder must choose among all possible transmitted codewords the codeword $\boldsymbol{c}$ that that minimizes the **Hamming distance** between $\boldsymbol{c}$ and $\bar{\boldsymbol{y}}$.

7.4.3 Soft-Decision Decoding

For soft-decision decoding, we start again from the ML rule (7.3) and proceed as

$$\begin{aligned}\hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \prod_{i=1}^{n} p(y_i|c_i)\\ &\overset{(a)}{=} \arg\max_{\boldsymbol{c}\in\mathcal{C}} \ln \prod_{i=1}^{n} p(y_i|c_i)\\ &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} \ln p(y_i|c_i), \qquad (7.7)\end{aligned}$$

where (a) follows because $\ln(\cdot)$ is a monotonically increasing function.

The channel law for the Gaussian channel is given in (5.2). Using this and $x_i = (-1)^{c_i}$ in (7.7),

$$\begin{aligned}\hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} \ln\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i-(-1)^{c_i})^2}{2\sigma^2}}\right)\\ &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} \left(\ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(y_i-(-1)^{c_i})^2}{2\sigma^2}\right)\\ &\overset{(b)}{=} \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} \frac{-(y_i-(-1)^{c_i})^2}{2\sigma^2}\\ &\overset{(c)}{=} \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} \frac{(y_i-(-1)^{c_i})^2}{2\sigma^2}\\ &\overset{(d)}{=} \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} (y_i-(-1)^{c_i})^2, \qquad (7.8)\end{aligned}$$

where in (b), we exploit the fact that addition of a constant does not change the maximization, in (c) we use the fact that maximizing a function is equivalent to minimizing minus the function, and finally in (d) we used that multipliying by the constant $2\sigma^2$ does not change the minimization.

Note that

$$\begin{aligned}\sum_{i=1}^{n} (y_i-(-1)^{c_i})^2 &= \sum_{i=1}^{n} (y_i - x_i)^2\\ &= \|\boldsymbol{y}-\boldsymbol{x}\|^2\\ &= d_{\mathsf{E}}^2(\boldsymbol{x},\boldsymbol{y}).\end{aligned}$$

Using this in (7.8) we can rewrite the ML decoding rule as

$$\begin{aligned}\hat{\boldsymbol{c}} &= \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{E}}^2(\boldsymbol{x},\boldsymbol{y})\\ &= \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{E}}(\boldsymbol{x},\boldsymbol{y}). \qquad (7.9)\end{aligned}$$

Therefore, the ML decoder consists in choosing among all possible transmitted codewords the codeword $\boldsymbol{c}$ that **minimizes the Euclidean distance** between the corresponding BPSK-modulated sequence $\boldsymbol{x}$ and $\boldsymbol{y}$.

We can also further simplify the expression (7.8) as follows,

$$
\begin{aligned}
\hat{\boldsymbol{c}} &= \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} (y_i - (-1)^{c_i})^2 \\
&= \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} (y_i^2 + (-1)^{2c_i} - 2y_i(-1)^{c_i}) \\
&= \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} (y_i^2 + 1 - 2y_i(-1)^{c_i}) \\
&\overset{(e)}{=} \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} (-2y_i(-1)^{c_i}) \\
&\overset{(f)}{=} \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{n} y_i(-1)^{c_i}, \qquad (7.10)
\end{aligned}
$$

where in (e) we used the fact that addition of y_i^2+1 does not change the maximization and in (f) we replaced minimization by maximization of the negation. The ML rule in the form of (7.10) will be useful in Chapter 9.

7.4.4 Comparison Between Soft-Decision Decoding and Hard-Decision Decoding

Obviously, since hard-decision decoding does not exploit all information at the output of the channel, i.e., we throw away some information, we would expect hard-decision decoding to perform worse than soft-decision decoding. This is clarified in the following example.

Example 7.4 Assume that we transmit the data bit $u = 0$ using the 3-repetition code of Example 7.1. $u = 0$ is encoded onto codeword $\boldsymbol{c} = (0,0,0)$, which is modulated onto $\boldsymbol{x} = (+1,+1,+1)$. After transmission over the AWGN channel, we receive $\boldsymbol{y} = (-0.2,+1.1,-0.7)$. We can see that two sign changes occurred during transmission, hence the hard-detected sequence is $\bar{\boldsymbol{y}} = (1,0,1)$. An ML hard-decision decoder using exclusively $\bar{\boldsymbol{y}}$ will assume that the codeword $\hat{\boldsymbol{c}} = (1,1,1)$ was transmitted. However, when we carry out ML soft-decision decoding (see (7.10)), for codewords $(0,0,0)$ and $(1,1,1)$ we get

$$
\begin{aligned}
(0,0,0): &\qquad \sum_{i=1}^{3} y_i(-1)^0 = +0.2 \\
(1,1,1): &\qquad \sum_{i=1}^{3} y_i(-1)^1 = -0.2
\end{aligned}
$$

We can see that the sum is maximized for $(0,0,0)$, thus the ML soft-decision decoder (correctly) decides for $\hat{\boldsymbol{c}} = (0,0,0)$ and hence $\hat{u} = 0$.

The example above highlights that hard-decision decoding incurs a performance penalty with respect to soft-decision decoding. On the other hand, hard-decision decoding is usually significantly less complex than soft-decision decoding.

In Figure 7.3, we plot the mutual information curves for BPSK transmission over the AWGN channel with soft-decision decoding and hard-decision decoding, together with the channel capacity curve. We observe that the mutual information for soft-decision decoding is close to the capacity for low spectral efficiencies, but it diverges for medium-to-high spectral efficiencies. It is also seen in the figure that hard-decision decoding results in a loss of 1 to 2 dB with respect to soft-decision decoding, depending on the spectral efficiency.

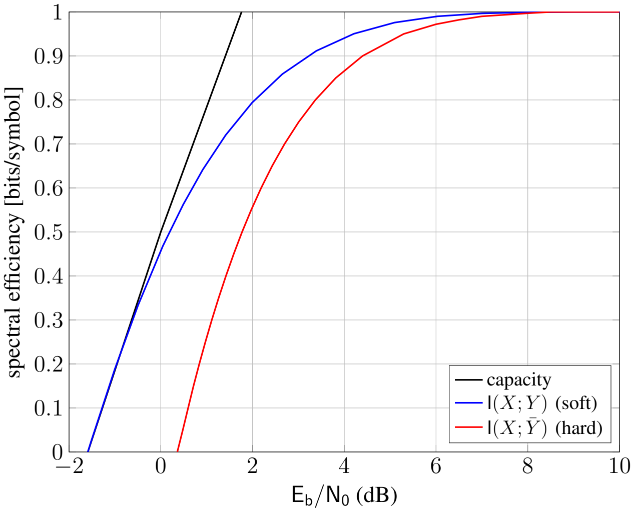


Figure 7.3: Mutual information curves for BPSK modulation and soft-decision decoding and hard-decision decoding, together with the channel capacity curve.

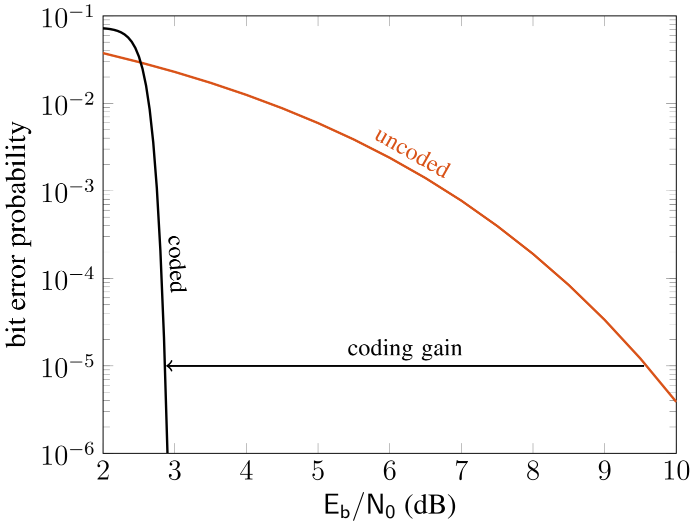


Figure 7.4: The advantage of using coding: the required $\mathsf{E_b/N_0}$ to achieve a given bit error probability is reduced.

For example, for $R = 0.5$ bits per symbol, hard-decision decoding performs 1.75 dB worse than soft-decision decoding.

7.5 The Advantage of Using Coding: Coding Gain

The advantage of coded transmission over uncoded transmission is usually measured in terms of **coding gain**, defined as the difference (in decibels) in the required $\mathsf{E_b/N_0}$ to achieve a given probability of error. The coding gain is shown qualitatively in Figure 7.4, where we plot the bit error probability P_b versus $\mathsf{E_b/N_0}$ for uncoded and coded transmission using BPSK. The coding gain in the figure may seem huge (9.8 dB!), but modern codes can yield even higher gains. As can be observed in the figure, the coding gain depends on the value of P_{b}. Furthermore, for low values of $\mathsf{E_b}/\mathsf{N_0}$ there is typically a crossing between the uncoded and the coded curves, meaning that uncoded transmission performs better. In other words, there is a limit to what a code can do in terms of error correction. This is due to the fact that, if the energy per information bit $\mathsf{E_b}$ is kept constant, coding decreases the energy per transmitted symbol to $\mathsf{E_s} = \mathsf{E_b} R_{\mathsf{c}}$ (note that for uncoded transmission $\mathsf{E_s} = \mathsf{E_b}$). If the channel is very noisy, coding cannot compensate for the loss in energy per transmitted symbol.

Chapter 8

Linear Block Codes

In this chapter, we discuss in more detail one of the most important classes of codes, namely linear block codes, already briefly introduced in Section 7.3. We introduce important concepts of linear block codes such as the generator matrix and the parity-check matrix, and discuss their error correcting capabilities and decoding over the BSC.

For block codes, the encoding is memoryless, i.e., a given block of k information bits $\boldsymbol{u}$ is always encoded to the same codeword of n coded bits $\boldsymbol{c}$, which depends exclusively on the k information bits at a given time instant and not on previous bits. Classical linear block codes such as Reed-Solomon and BCH codes are currently used in multiple applications, including data-storage systems (e.g., hard-disk drives and DVDs), satellite communications, fiber-optic communications, and distributed storage (e.g., Facebook's Hadoop distributed file system and Microsoft's Windows Azure cloud system).

8.1 Generator Matrix

As already discussed in Section 7.3, a $\mathcal{C}(n,k)$ linear block code is a k-dimensional subspace of the n-dimensional vector space of all binary vectors of length n, i.e., $\mathcal{C} \subset \{0,1\}^n$. Therefore, as for any subspace, we can find k linearly independent basis vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k \in \{0,1\}^n$ that span $\mathcal{C}$. In other words, every codeword in $\mathcal{C}$ is a linear combination of these k basis vectors. Then, the codeword $\boldsymbol{c} = (c_1, \ldots, c_n)$ for the message $\boldsymbol{u} = (u_1, \ldots, u_k)$ can be expressed as a linear combination of the k basis vectors $\{\boldsymbol{g}_i\}$ as

$$\boldsymbol{c} = \sum_{i=1}^{k} u_i \boldsymbol{g}_i = u_1 \boldsymbol{g}_1 + \ldots u_k \boldsymbol{g}_k. \tag{8.1}$$

Note that the basis vectors $\{\boldsymbol{g}_i\}$ are also codewords of $\mathcal{C}$. This is easily seen by letting the information word $\boldsymbol{u}$ contain a single one in position i.

We can rewrite (8.1) in matrix form as

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}, \tag{8.2}$$

where $\boldsymbol{G}$ is a $k \times n$ binary matrix with rows $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$. Since $\boldsymbol{G}$ spans (i.e., generates) the code $\mathcal{C}$, it is usually referred to as the **generator matrix** of the code.

It is important to realize that both the code as well as the encoder (i.e., the mapping of information words to codewords), are completely specified by the generator matrix $\boldsymbol{G}$. We can formally define a code through its generator matrix as

$$\boxed{\mathcal{C} \triangleq \{\boldsymbol{c} : \boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}\}.} \tag{8.3}$$

For each subspace, there are several bases that generate the same subspace. Correspondingly, there are several generator matrices that generate the same code $\mathcal{C}$. Each generator matrix $\boldsymbol{G}$ corresponds to a different encoder, i.e., to a different mapping between information words and codewords (see Definition 7.2). Two encoders, or equivalently two generator matrices, that generate the same code are called **equivalent encoders**.

From basic linear algebra, any linear operation on the basis vectors leads to another basis that generates the same subspace. Applying this principle, for any code $\mathcal{C}$ it is always possible to find a generator matrix in the form

$$\boxed{\boldsymbol{G}_{\mathsf{s}} = (\boldsymbol{I}_k \quad \boldsymbol{P}),} \tag{8.4}$$

where $\boldsymbol{I}_k$ is a $k \times k$ identity matrix and $\boldsymbol{P}$ is a $k \times (n-k)$ matrix. A generator matrix in the form of (8.4) is called a **systematic generator matrix**, and the resulting code is called a **systematic code**. The codewords of a systematic code can be written as

$$\boldsymbol{c} = (c_1 \ldots, c_n) = \boldsymbol{u}\boldsymbol{G}_{\mathsf{s}} = (u_1, , \ldots, u_k, p_1, \ldots p_{n-k}) = (\boldsymbol{u} \ \boldsymbol{p}),$$

i.e., the first k bits of the codeword are replicas of the information bits. The code bits $p_1, p_2, \ldots, p_{n-k}$ are referred to as the **parity bits** of the code.

We give a formal definition of a systematic code in the following.

Definition 8.1 (Systematic code) A *systematic code* is a linear block code that contains the information word $\boldsymbol{u}$ as verbatim copy in $\boldsymbol{c}$.

Systematic codes are an important subclass of linear codes, since encoding and decoding are easier.

Example 8.1 ((3, 1) Repetition code) We consider again the 3-repetition code of Example 7.1 and the encoder defined in the example, i.e., $0 \to (0,0,0)$ and $1 \to (1,1,1)$. The corresponding generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

It is easy to see that the code is systematic.

Example 8.2 ((3, 2) Parity-check code) The parity-check code with parameters $(n, k) = (3, 2)$ consisting of $2^k = 4$ codewords of length 3 bits is

$$\mathcal{C}_{\mathsf{check}}(n = 3, k = 2) = \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

The parity-check code is such that the code bit c_3 is a parity-check of the first two code bits, i.e., $c_3 = c_1 + c_2$. One may consider the following (systematic) encoder,

$$\begin{aligned}
\boldsymbol{u} = (0,0) &\quad\to\quad \boldsymbol{c}_1 = (0,0,0)\\
\boldsymbol{u} = (0,1) &\quad\to\quad \boldsymbol{c}_2 = (0,1,1)\\
\boldsymbol{u} = (1,0) &\quad\to\quad \boldsymbol{c}_3 = (1,0,1)\\
\boldsymbol{u} = (1,1) &\quad\to\quad \boldsymbol{c}_4 = (1,1,0).
\end{aligned}$$

The code has rate $R_{\mathsf{c}} = 2/3$ and is a systematic code. The corresponding generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}. \tag{8.5}$$

It is easy to see that $d_{\mathsf{min}}(\mathcal{C}_{\mathsf{check}}) = 2$.

Example 8.3 ((7, 4) Hamming code) The first error correcting codes were introduced by Richard Hamming in 1950 and are now known as Hamming codes. Hamming codes have parameters $n = 2^r - 1$, $k = 2^r - r - 1$ with $r \geq 3$, and $d_{\mathsf{min}} = 3$. The smallest Hamming code is the $(7, 4)$ Hamming code, consisting of $2^k = 16$ codewords of length 7 bits,

$$\begin{aligned}\mathcal{C}_{\mathsf{Hamm}}(n = 7, k = 4) = \{&(0,0,0,0,0,0,0),(0,0,0,1,0,1,1),(0,0,1,0,1,1,0),(0,0,1,1,1,0,1),\\ &(0,1,0,0,1,1,1),(0,1,0,1,1,0,0),(0,1,1,0,0,0,1),(0,1,1,1,0,1,0)\\ &(1,0,0,0,1,0,1),(1,0,0,1,1,1,0),(1,0,1,0,0,1,1),(1,0,1,1,0,0,0)\\ &(1,1,0,0,0,1,0),(1,1,0,1,0,0,1),(1,1,1,0,1,0,0),(1,1,1,1,1,1,1)\}.\end{aligned}$$

In its systematic form, the encoder of the $(7, 4)$ Hamming code is given by

$$\begin{aligned}
\boldsymbol{u} &= (0,0,0,0) & &\rightarrow & \boldsymbol{c}_1 &= (0,0,0,0,0,0,0) & \qquad \boldsymbol{u} &= (1,0,0,0) & &\rightarrow & \boldsymbol{c}_9 &= (1,0,0,0,1,0,1)\\
\boldsymbol{u} &= (0,0,0,1) & &\rightarrow & \boldsymbol{c}_2 &= (0,0,0,1,0,1,1) & \qquad \boldsymbol{u} &= (1,0,0,1) & &\rightarrow & \boldsymbol{c}_{10} &= (1,0,0,1,1,1,0)\\
\boldsymbol{u} &= (0,0,1,0) & &\rightarrow & \boldsymbol{c}_3 &= (0,0,1,0,1,1,0) & \qquad \boldsymbol{u} &= (1,0,1,0) & &\rightarrow & \boldsymbol{c}_{11} &= (1,0,1,0,0,1,1)\\
\boldsymbol{u} &= (0,0,1,1) & &\rightarrow & \boldsymbol{c}_4 &= (0,0,1,1,1,0,1) & \qquad \boldsymbol{u} &= (1,0,1,1) & &\rightarrow & \boldsymbol{c}_{12} &= (1,0,1,1,0,0,0)\\
\boldsymbol{u} &= (0,1,0,0) & &\rightarrow & \boldsymbol{c}_5 &= (0,1,0,0,1,1,1) & \qquad \boldsymbol{u} &= (1,1,0,0) & &\rightarrow & \boldsymbol{c}_{13} &= (1,1,0,0,0,1,0)\\
\boldsymbol{u} &= (0,1,0,1) & &\rightarrow & \boldsymbol{c}_6 &= (0,1,0,1,1,0,0) & \qquad \boldsymbol{u} &= (1,1,0,1) & &\rightarrow & \boldsymbol{c}_{14} &= (1,1,0,1,0,0,1)\\
\boldsymbol{u} &= (0,1,1,0) & &\rightarrow & \boldsymbol{c}_7 &= (0,1,1,0,0,0,1) & \qquad \boldsymbol{u} &= (1,1,1,0) & &\rightarrow & \boldsymbol{c}_{15} &= (1,1,1,0,1,0,0)\\
\boldsymbol{u} &= (0,1,1,1) & &\rightarrow & \boldsymbol{c}_8 &= (0,1,1,1,0,1,0) & \qquad \boldsymbol{u} &= (1,1,1,1) & &\rightarrow & \boldsymbol{c}_{16} &= (1,1,1,1,1,1,1).
\end{aligned}$$

Equivalently, the $(7, 4)$ Hamming code is defined by the relations

$$\begin{aligned} c_i &= u_i, \quad i = 1, 2, 3, 4\\ c_5 &= u_1 + u_2 + u_3\\ c_6 &= u_2 + u_3 + u_4\\ c_7 &= u_1 + u_2 + u_4. \end{aligned}$$

The corresponding generator matrix is

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

The $(7, 4)$Â Hamming code rate has rate $R_{\mathsf{c}} = 4/7$. It is easy to verify that $d_{\mathsf{min}}(\mathcal{C}_{\mathsf{Hamm}}) = 3$.

8.2 Parity-Check Matrix

Linear block codes can also be described in an alternative way. Since a binary (n, k) linear block code $\mathcal{C}$ is a k-dimensional subspace of $\{0,1\}^n$, its **null** (or **dual**) space, i.e., the set of all binary words of length n that are orthogonal[1] to every element in $\mathcal{C}$, is an $(n-k)$-dimensional subspace of $\{0,1\}^n$. We denote the null space of $\mathcal{C}$ by $\mathcal{C}_\perp$, and we can write

$$\mathcal{C}_\perp = \{\tilde{\boldsymbol{c}} : \langle \tilde{\boldsymbol{c}}, \boldsymbol{c} \rangle = 0 \text{ for all } \boldsymbol{c} \in \mathcal{C}\}. \tag{8.6}$$

Note that $\mathcal{C}_\perp$ is a binary $(n, n-k)$ linear block code, usually referred to as the **dual code** of $\mathcal{C}$. Let $\boldsymbol{H}$ be a generator matrix of the code $\mathcal{C}_\perp$, of dimensions $(n-k) \times n$. The rows of $\boldsymbol{H}$, $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n-k}$, are linearly independent codewords of $\mathcal{C}_\perp$, i.e., $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n-k} \in \mathcal{C}_\perp$ form a basis that spans $\mathcal{C}_\perp$, and every codeword in $\mathcal{C}_\perp$ is a linear combination of these $n-k$ codewords. Since every codeword $\boldsymbol{c} \in \mathcal{C}$ is orthogonal to every codeword $\tilde{\boldsymbol{c}} \in \mathcal{C}_\perp$, it follows

$$\boxed{\boldsymbol{c}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}_{n-k}.} \tag{8.7}$$

It also follows

$$\boxed{\boldsymbol{G}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}_{k \times (n-k)}.} \tag{8.8}$$

For ease of notation, in the following we will drop the subindex from $\boldsymbol{0}_{n-k}$ and $\boldsymbol{0}_{k \times (n-k)}$. The dimensions of the vector (or matrix) $\boldsymbol{0}$ should be understood from the context.

Note that (8.7) is satisfied *if and only if* $\boldsymbol{c} \in \mathcal{C}$. Therefore, we can define a code $\mathcal{C}$ through the generator matrix of its dual code $\mathcal{C}_\perp$: A vector $\boldsymbol{c}$ is a codeword of $\mathcal{C}$ if and only if $\boldsymbol{c}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}$. We formalize this in the following definition.

Definition 8.2 A binary vector $\boldsymbol{c}$ is a codeword of $\mathcal{C}$ if an only if $\boldsymbol{c}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}$, i.e., the code $\mathcal{C}$ is defined as the null space of $\boldsymbol{H}$,

$$\boxed{\mathcal{C} = \{\boldsymbol{c} : \boldsymbol{c}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}\}.}$$

Therefore, a linear block code is uniquely specified by $\boldsymbol{G}$ and $\boldsymbol{H}$. Usually, the generator matrix is used for encoding, while the decoding is based on $\boldsymbol{H}$, as we will see in the next section.

A way to interpret (8.7) is by noticing that $\boldsymbol{c}\boldsymbol{H}^\mathsf{T} = \boldsymbol{0}$ is a set of $n-k$ linearly independent equations

$$\begin{array}{ccccccccc}
h_{11}c_1 & + & h_{12}c_2 & + & \ldots & + & h_{1n}c_n & = & 0 \\
h_{21}c_1 & + & h_{22}c_2 & + & \ldots & + & h_{2n}c_n & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
h_{(n-k)1}c_1 & + & h_{(n-k)2}c_2 & + & \ldots & + & h_{(n-k)n}c_n & = & 0
\end{array}$$

where h_{ij} is the entry of the matrix $\boldsymbol{H}$ at row i and column j. These equations are known as **parity-check equations** and are the equations that each codeword must satisfy. As a result, the matrix $\boldsymbol{H}$ is commonly referred to as the **parity-check matrix** of the code $\mathcal{C}$.

The parity-check matrix of a code is not unique, i.e., we can find several parity-check matrices that generate the same code. Performing elementary row operations, possibly in combination with column permutations, leads to another parity-check matrix $\tilde{\boldsymbol{H}}$ that generates the same code, i.e., $\boldsymbol{c}\tilde{\boldsymbol{H}}^\mathsf{T} = \boldsymbol{0}$.

[1] Two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of length n are orthogonal if their inner product is zero, i.e., $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}\boldsymbol{b}^\mathsf{T} = (a_1 b_1 + \ldots, a_n b_n) = 0$.

If the generator matrix of an (n,k) code is given in systematic form, $\boldsymbol{G}_{\mathsf{s}} = (\boldsymbol{I}_k \;\; \boldsymbol{P})$ (see (8.4)), its corresponding parity-check matrix in systematic form is given by

$$\boxed{\boldsymbol{H}_{\mathsf{s}} = \left(\boldsymbol{P}^{\mathsf{T}} \;\; \boldsymbol{I}_{n-k}\right).} \tag{8.9}$$

It is easy to see that $\boldsymbol{G}_{\mathsf{s}}\boldsymbol{H}_{\mathsf{s}}^{\mathsf{T}} = \boldsymbol{0}$.

Example 8.4 ((3, 2) Parity-check code) For the parity-check code in Example 8.2 with generator matrix in systematic form given in (8.5), the corresponding parity-check matrix in systematic form can be obtained using (8.9) as

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

Note that this matrix is the generator matrix of the $(3,1)$ repetition code (see Example 8.1)! Therefore, the $(3,1)$ repetition code is the dual code of the $(3,2)$ parity-check code, and vice versa.

8.3 Decoding of Linear Block Codes: Syndrome-Based Decoding

In this section, we discuss optimum decoding of linear block codes for hard-decision decoding. We have seen that for hard-decision decoding, transmission over the AWGN channel can be modeled as a BSC. Consider an (n,k) linear block code $\mathcal{C}$ with parity-check matrix $\boldsymbol{H}$. Assume that codeword $\boldsymbol{c} = (c_1,\ldots,c_n) \in \mathcal{C}$ is transmitted over the BSC and let $\bar{\boldsymbol{y}} = (\bar{y}_1,\ldots,\bar{y}_n)$ be the hard-decision received vector. Since the channel is noisy, the received vector $\bar{\boldsymbol{y}}$ may be different than the transmitted codeword $\boldsymbol{c}$, i.e., $\bar{\boldsymbol{y}}$ and $\boldsymbol{c}$ may differ in some positions. We can write $\bar{\boldsymbol{y}}$ as

$$\bar{\boldsymbol{y}} = \boldsymbol{c} + \boldsymbol{e}, \tag{8.10}$$

where the entries of the vector $\boldsymbol{e} = (e_1,\ldots,e_n)$ are one in the positions where transmission errors have occurred and zero otherwise. Vector $\boldsymbol{e}$ is usually referred to as the **error vector** or the **error pattern**.

At the receiver, neither the transmitted codeword $\boldsymbol{c}$ nor the error pattern $\boldsymbol{e}$ are known. As discussed in Section 7.4.2, the ML decoder decodes $\bar{\boldsymbol{y}}$ onto the codeword $\boldsymbol{c}$ that is closer (in Hamming distance) to $\bar{\boldsymbol{y}}$, i.e., it selects

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}}). \tag{8.11}$$

The ML decoding rule (8.11) is equivalent to finding the error pattern $\boldsymbol{e}$ with smallest Hamming weight that we need to add to $\bar{\boldsymbol{y}}$ to obtain a valid codeword. In fact, from (8.11) we can write

$$\begin{aligned}\hat{\boldsymbol{c}} &= \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c},\bar{\boldsymbol{y}})\\ &\overset{(a)}{=} \arg\min_{\boldsymbol{c}\in\mathcal{C}} w_{\mathsf{H}}(\boldsymbol{c}+\bar{\boldsymbol{y}})\\ &\overset{(b)}{=} \arg\min_{\substack{\boldsymbol{e}\\ \bar{\boldsymbol{y}}+\boldsymbol{e}=\boldsymbol{c}\in\mathcal{C}}} w_{\mathsf{H}}(\boldsymbol{e})\end{aligned} \tag{8.12}$$

where in (a) we used (7.1), and in (b) we used (8.10).

We use this fact to introduce an efficient way to decode linear block codes, referred to as **syndrome decoding**, which exploits the linearity of the code. Using the parity-check matrix $\boldsymbol{H}$, we have

$$\begin{aligned}&\boxed{\bar{\boldsymbol{y}}\boldsymbol{H}^{\mathsf{T}} = (\boldsymbol{c}+\boldsymbol{e})\boldsymbol{H}^{\mathsf{T}}}\\ &\boxed{\overset{(a)}{=} \boldsymbol{e}\boldsymbol{H}^{\mathsf{T}}}\end{aligned} \tag{8.13}$$

where in (a) we used the fact that $\boldsymbol{c}\boldsymbol{H}^{\mathsf{T}} = \boldsymbol{0}\ \forall \boldsymbol{c} \in \mathcal{C}$. We define the vector $\boldsymbol{s}$ as

$$\boldsymbol{s} \triangleq \bar{\boldsymbol{y}}\boldsymbol{H}^{\mathsf{T}} = \boldsymbol{e}\boldsymbol{H}^{\mathsf{T}}. \tag{8.14}$$

The vector $\boldsymbol{s} = (s_1, \ldots, s_{n-k})$ is of length $n-k$ and is commonly called the **syndrome**. Note that the received vector $\bar{\boldsymbol{y}}$ is a codeword, i.e., $\bar{\boldsymbol{y}} \in \mathcal{C}$, if and only if the syndrome is zero (recall the definition of a linear block code as the null space of $\boldsymbol{H}$, see Definition 8.2). If the received vector $\bar{\boldsymbol{y}}$ is a codeword, i.e., $\boldsymbol{s} = \boldsymbol{0}$, then the most-likely transmitted codeword is $\boldsymbol{c} = \bar{\boldsymbol{y}}$ (the one closest in Hamming distance, in this case $d_{\mathsf{min}}(\boldsymbol{c}, \bar{\boldsymbol{y}}) = 0$).

The syndrome $\boldsymbol{s}$ is the all-zero vector in two situations: (i) if $\boldsymbol{e} = \boldsymbol{0}$, i.e., the channel has introduced no errors, or (ii) the error pattern introduced by the channel is such that $\bar{\boldsymbol{y}} = \boldsymbol{c} + \boldsymbol{e} \in \mathcal{C}$ but $\bar{\boldsymbol{y}} \neq \boldsymbol{c}$, i.e., $\bar{\boldsymbol{y}}$ is another codeword, different from the transmitted one. By definition of a linear code (see Definition 7.6), $\bar{\boldsymbol{y}} = (\boldsymbol{c} + \boldsymbol{e}) \in \mathcal{C}$ and $\bar{\boldsymbol{y}} \neq \boldsymbol{c}$ occurs if and only if $\boldsymbol{e}$ is a nonzero codeword, i.e., $\boldsymbol{e} \in \mathcal{C}$ and $\boldsymbol{e} \neq \boldsymbol{0}$. If this happens, the decoder will decide erroneously that the transmitted codeword was $\bar{\boldsymbol{y}}$. An error pattern of this type is called an **undetectable error pattern**, because it is impossible to correct by the decoder. Obviously, there are $2^k - 1$ such undetectable error patterns, corresponding to the number of codewords of $\mathcal{C}$ except the all-zero codeword.

Now let's consider the case where the syndrome is not the all-zero vector. Since there are 2^{n-k} possible syndromes $\boldsymbol{s} = (s_1, \ldots, s_{n-k})$, i.e., all binary vectors of length $n-k$, there are

$$\frac{2^n}{2^{n-k}} = 2^k$$

received vectors $\bar{\boldsymbol{y}}$ that generate the same syndrome. Equivalently, there are 2^k error patterns $\boldsymbol{e}$ that generate the same syndrome, or in other words, 2^k solutions to the equation (8.14). For the BSC, the most likely solution to the equation (8.14) is the vector $\boldsymbol{e}$ with smallest Hamming weight. In fact, if two vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ with $w_{\mathsf{H}}(\boldsymbol{e}_1) < w_{\mathsf{H}}(\boldsymbol{e}_2)$Â are valid solutions to (8.14), $\boldsymbol{e}_1$ is the most likely solution because it is more likely that the BSC has generated the error vector $\boldsymbol{e}_1$ than the error vector $\boldsymbol{e}_2$.

Therefore, the ML decoder can be interpreted as follows: For a received vector $\bar{\boldsymbol{y}}$ that generates the syndrome $\boldsymbol{s}$, among all possible 2^k error patterns that generate $\boldsymbol{s}$ find the one with smallest Hamming weight, which we denote by $\boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s})$. Then, assume that this is the error introduced by the channel and decode $\bar{\boldsymbol{y}}$ onto

$$\hat{\boldsymbol{c}} = \bar{\boldsymbol{y}} + \boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s}). \tag{8.15}$$

Note that this decoder performs the ML rule in (8.12): It selects among all possible error vectors $\boldsymbol{e}$ the one with smallest Hamming weight that added to $\bar{\boldsymbol{y}}$ yields a valid codeword. We simplify the search of $\boldsymbol{e}_{\mathsf{min}}$ by exploiting (8.13). Since this decoding procedure is based on the syndrome, it is referred to as **syndrome decoding**.

ML syndrome decoding can be implemented efficiently (for relatively short codes for which the number of syndromes, 2^{n-k}, is reasonable) by using a **decoding table** that associates to each syndrome $\boldsymbol{s}$ the error pattern $\boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s})$ with smallest weight that generates it. As we have seen, this is the most likely error pattern to occur that leads to this syndrome. The error patterns listed in the decoding table are hence the **correctable** error patterns. On the other hand, if the channel introduces an error pattern different from those in the decoding table, a decoding error will occur.

To summarize, the decoding of a received vector $\bar{\boldsymbol{y}}$ is then carried out in three steps:

1. Compute the syndrome of $\bar{\boldsymbol{y}}$, $\boldsymbol{s} = \bar{\boldsymbol{y}}\boldsymbol{H}^{\mathsf{T}}$.

2. Find in the decoding table the error pattern $\boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s})$ whose syndrome is equal to $\boldsymbol{s}$. $\boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s})$ is assumed to be the error pattern introduced by the channel.

3. Decode $\bar{\boldsymbol{y}}$ onto the codeword $\hat{\boldsymbol{c}} = \bar{\boldsymbol{y}} + \boldsymbol{e}_{\mathsf{min}}(\boldsymbol{s})$.

Example 8.5 ((7,4) Hamming code) A possible parity-check matrix for the $(7,4)$ Hamming code of Example 8.3, obtained using (8.9), is

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{8.16}$$

There are $2^3 = 8$ possible syndromes, namely all binary vectors of length 3. We can build the decoding table assigning to each syndrome $\boldsymbol{s}$Â the most likely error pattern that generates it, i.e., the error pattern of smallest Hamming weight that generates $\boldsymbol{s}$. The decoding table for the $(7,4)$ Hamming code is given below.

$\boldsymbol{s}$	$\boldsymbol{e}_{\mathsf{min}}$
(0,0,0)	(0,0,0,0,0,0,0)
(0,0,1)	(0,0,0,0,0,0,1)
(0,1,0)	(0,0,0,0,0,1,0)
(0,1,1)	(0,0,0,1,0,0,0)
(1,0,0)	(0,0,0,0,1,0,0)
(1,0,1)	(1,0,0,0,0,0,0)
(1,1,0)	(0,0,1,0,0,0,0)
(1,1,1)	(0,1,0,0,0,0,0)

Assume that we receive $\bar{\boldsymbol{y}} = (1,1,0,0,0,0,0)$. We first compute the syndrome, $\boldsymbol{s} = \bar{\boldsymbol{y}}\boldsymbol{H}^{\mathsf{T}} = (0,1,0)$. We then find in the decoding table that $\boldsymbol{e}_{\mathsf{min}}(0,1,0) = (0,0,0,0,0,1,0)$ is the most likely error pattern to have generated this syndrome. Therefore, the ML decision is to decode $\bar{\boldsymbol{y}}$ onto $\hat{\boldsymbol{c}} = \bar{\boldsymbol{y}} + \boldsymbol{e}_{\mathsf{min}}(0,1,0) = (1,1,0,0,0,1,0)$, which corresponds to $\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{min}}(\bar{\boldsymbol{y}}, \boldsymbol{c})$. Note that, for example, the error pattern $\boldsymbol{e} = (1,1,0,0,0,0,0)$ generates the same syndrome, i.e., $\boldsymbol{s} = \boldsymbol{e}\boldsymbol{H}^{\mathsf{T}} = (0,1,0)$. Therefore, if the actual error introduced by the channel was $\boldsymbol{e} = (1,1,0,0,0,0,0)$, we would have decoded incorrectly onto $\hat{\boldsymbol{c}} = (1,1,0,0,0,1,0)$.

8.4 Error Correction and Error Detection Capabilities of Block Codes over the BSC

We have already discussed in Section 7.2 that the error correction and error detection capabilities of a code are directly related to its minimum Hamming distance. Let us formalize this in the following theorems.

Theorem 8.1 (Error correction capability) *For transmission over the BSC, a block code with minimum Hamming distance d_{min} can correct all error patterns with*

$$t = \left\lfloor \frac{d_{\mathsf{min}} - 1}{2} \right\rfloor$$

or fewer errors.[2]

Proof: Interpreting the ML decoder geometrically, we can define around each codeword a decision region which contains all received sequences that are closer to this codeword than to any other codeword. Then, given a transmitted codeword, the maximum number of errors which keeps the received sequence within the decision region corresponding to the transmitted codeword is $\lfloor \frac{d_{\mathsf{min}}-1}{2} \rfloor$, in which case the error can be corrected. ■

Parameter t is usually referred to as the **error correction capability** of the code, and a code that can correct up to t errors is called a **t-error correcting code**.

In this course, we mainly focus on **error correction**. However, we can also consider coding for **error detection**, in which case the role of the decoder is to detect possible transmission errors, without attempting to correct them.

Theorem 8.2 (Error detection capability) *For transmission over the BSC, a code with minimum Hamming distance d_{min} can detect all error patterns with*

$$d = d_{\mathsf{min}} - 1$$

or fewer errors.

Proof: Consider again the geometry of the code. A codeword is a vector in an n-dimensional space. The minimum distance between any two codewords is d_{min}. Therefore, if the error vector has Hamming weight $w_{\mathsf{H}}(\boldsymbol{e}) \leq d_{\mathsf{min}} - 1$, the resulting received vector $\bar{\boldsymbol{y}} = \boldsymbol{c} + \boldsymbol{e}$ cannot be a codeword, hence the error is detectable. ■

Example 8.6 ((7, 4) Hamming code) It is easy to verify that the $(7,4)$ Hamming code (see Example 8.3) has $d_{\mathsf{min}} = 3$. Therefore, it can correct *all* error patterns of Hamming weight up to 1 and detect *all* error patterns of Hamming weight up to 2. Codes that can correct single errors, as the $(7,4)$ Hamming code, are usually referred to as single-error correcting codes.

[2] Here, $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x. For example $\lfloor 1.3 \rfloor = 1$, $\lfloor 1.8 \rfloor = 1$, and $\lfloor 2.0 \rfloor = 2$.

Chapter 9

Convolutional Codes

In this chapter, we discuss convolutional codes, an important class of codes introduced by Peter Elias in 1955. Since the 1970s, convolutional codes have been widely used in wireless networks, satellite and spacecraft links, and terrestrial broadcast communications. The key enabler of the popularity of convolutional codes is their associated low-complexity ML decoding algorithm, the well-known **Viterbi decoding algorithm**. Furthermore, they yield excellent performance when concatenated with block codes (e.g., Reed-Solomon codes). Today, convolutional codes are still used as stand-alone codes in several communication standards, such as Ethernet cables, and are also used in concatenation with Reed-Solomon codes for NASA's free-space satellite communications. They are also the main ingredients of turbo-like codes, one of the state-of-the-art coding schemes included in most of the current communication standards, which we will introduce in the next chapter.

In the following sections, we discuss the main principles of convolutional codes. We introduce a useful graphical representation of convolutional codes, the so-called trellis diagram, and describe in detail their ML decoding based on the Viterbi algorithm. Finally, we also discuss bounds on the error probability of convolutional codes and linear codes in general.

9.1 Main Principle

Convolutional codes are linear codes with a very distinct algebraic structure as compared to block codes. As we have seen in the previous chapter, for block codes the information sequence is parsed into blocks of k bits, $\boldsymbol{u}$, and each block is encoded independently onto a block (codeword) of n code bits, $\boldsymbol{c}$, as $\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}$. The codeword $\mathbf{c}$ depends only on the information bits $\boldsymbol{u}$. In other words, a block of k information bits $\boldsymbol{u}$ is always encoded into the same codeword $\boldsymbol{c}$. In contrast, convolutional codes introduce **memory**: The n code bits that the convolutional encoder generates in correspondence to the k information bits at its input depend also on previous information bits. Convolutional codes are **stream-oriented** in nature; we can see a convolutional encoder as a device that encodes a potentially infinitely long sequence of information bits into an infinitely long sequence of code bits. For this reason, in convolutional coding sometimes we refer to codewords as code sequences, to emphasize that they may be of infinite length. The main principle of convolutional codes will become much clearer in the following.

Since the mathematical description of convolutional codes is significantly more involved than that of block codes, to introduce convolutional codes in a simple and accessible way we will follow the conventional approach

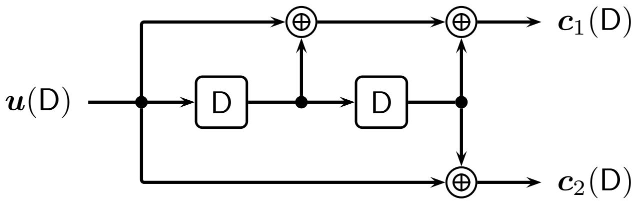

$\boldsymbol{u}(\mathsf{D})$
D
D
$\boldsymbol{c}_1(\mathsf{D})$
$\boldsymbol{c}_2(\mathsf{D})$

Figure 9.1: $R_{\mathsf{c}} = 1/2$ convolutional encoder with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \;\; 1 + \mathsf{D}^2)$. For each information bit, the encoder generates two code bits.

in basic coding theory books and introduce them by using a simple convolutional code as an example, the so-called **convolutional code archetype**.

The convolutional code archetype is a four-state, rate $R_{\mathsf{c}} = 1/2$ code whose encoder is depicted Figure 9.1. A convolutional encoder can be regarded as a **finite-state machine**. The information sequence at the input of the encoder, denoted by $\boldsymbol{u} = (u_1, u_2, \ldots)$, has potentially an infinite length. The encoder generates two sequences at its output, denoted by $\boldsymbol{c}_1 = (c_{1,1}, c_{1,2}, \ldots)$ and $\boldsymbol{c}_2 = (c_{2,1}, c_{2,2}, \ldots)$, respectively. At each time instant i, two coded bits, $c_{1,i}$ and $c_{2,i}$, are produced for each information bit u_i; hence, the code rate is $R_{\mathsf{c}} = 1/2$.

As for block codes, we define a convolutional code by the parameters (n, k). However, it is important to realize that while for block codes a code with parameters (n, k) is a code of length n and dimension k, i.e., the corresponding encoder encodes information words of length k into codewords of length n, the meaning of the parameters (n, k) is a bit different for convolutional codes. Here, (n, k) means that for each k input bits the convolutional encoder generates n output bits. The code rate is (as for block codes) $R_{\mathsf{c}} = k/n$. For the convolutional code resulting from the encoder in Figure 9.1, $(n, k) = (2, 1)$ and the code rate is $R_{\mathsf{c}} = k/n = 1/2$.[1]

The encoder of Figure 9.1 has two memory elements (represented by the squares with the legend D inside), which take values on $\{0, 1\}$. The meaning of D will become clear later. The number of memory elements of a convolutional encoder is called the **memory** of the encoder, which we will denote by ν. In general, all memory elements are initialized to zero before encoding starts. During the encoding process, the content of the memory elements changes. Each state of the memory is referred to as a **state** of the encoder and an encoder with memory ν has 2^ν states (all binary vectors of length ν). For the example in Figure 9.1, $\nu = 2$ hence the encoder has $2^\nu = 4$ possible states, namely $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$. We will denote the state of the encoder at time instant i by $\boldsymbol{s}_i \in \{0, 1\}^\nu$. Note that both the state of the encoder at time instant $i + 1$, $\boldsymbol{s}_{i+1}$, and the code bits at time instant i, $c_{1,i}$ and $c_{2,i}$, are deterministic functions of the encoder state $\boldsymbol{s}_i$ and the input bit u_i at time i. Hence, we will write $\boldsymbol{s}_{i+1}(\boldsymbol{s}_i, u_i)$, $c_{1,i}(\boldsymbol{s}_i, u_i)$, and $c_{2,i}(\boldsymbol{s}_i, u_i)$. In other words, the code bits at a given time instant i depend on the information bit at time i, u_i, and also on some previous information bits (the encoder has memory) through the state of the encoder.

The top and bottom branches of Figure 9.1 act as two discrete-time finite-impulse response filters with operations in the binary field. The top filter has impulse response $\boldsymbol{g}_1 = (1, 1, 1)$, while the bottom filter has impulse response $\boldsymbol{g}_2 = (1, 0, 1)$. The code sequences $\boldsymbol{c}_1 = (c_{1,1}, c_{1,2}, \ldots)$ and $\boldsymbol{c}_2 = (c_{2,1}, c_{2,2}, \ldots)$ can then be obtained as the convolution of the information sequence $\boldsymbol{u} = (u_1, u_2, \ldots)$ and the impulse responses $\boldsymbol{g}_1$ and

[1] As for block codes, rigorously speaking, the convolutional encoder defines the mapping between information sequences and code sequences, while the convolutional code is the set of codewords at the output of the encoder. However, for convolutional codes, with some abuse of language, code and encoder are used in a loose way, and sometimes one uses code to refer to the encoder too.

$\boldsymbol{g}_2$, i.e.,

$$\boldsymbol{c}_1 = \boldsymbol{u} * \boldsymbol{g}_1, \tag{9.1}$$

$$\boldsymbol{c}_2 = \boldsymbol{u} * \boldsymbol{g}_2, \tag{9.2}$$

where $*$ denotes convolution. This explains the name of convolutional codes.

In general, if the convolutional encoder has parameters $(n, 1)$, we will write

$$\boldsymbol{c}_j = \boldsymbol{u} * \boldsymbol{g}_j, \quad j = 1, \ldots, n. \tag{9.3}$$

9.1.1 D-Transform Notation

The fact that the input-output relationship of a convolutional encoder may be written as a convolution suggests the use of a polynomial-like notation, in which convolution reverts to multiplication. Let us define the power series

$$\begin{aligned}\boldsymbol{u}(\mathsf{D}) &= \sum_i u_i \mathsf{D}^i,\\ \boldsymbol{c}(\mathsf{D}) &= \sum_i c_i \mathsf{D}^i,\\ \boldsymbol{g}(\mathsf{D}) &= \sum_i g_i \mathsf{D}^i,\end{aligned}$$

which are usually referred to as **D-transforms**. Note that $\boldsymbol{u}$, $\boldsymbol{c}$, and $\boldsymbol{g}$ can be obtained from the coefficients of the corresponding D-transforms $\boldsymbol{u}(\mathsf{D})$, $\boldsymbol{c}(\mathsf{D})$, and $\boldsymbol{g}(\mathsf{D})$.

Example 9.1 Given $\boldsymbol{u} = (1, 0, 0, 1, 1, \ldots)$ the corresponding D-transform is

$$\boldsymbol{u}(\mathsf{D}) = 1 \cdot \mathsf{D}^0 + 0 \cdot \mathsf{D}^1 + 0 \cdot \mathsf{D}^2 + 1 \cdot \mathsf{D}^3 + 1 \cdot \mathsf{D}^4 = 1 + \mathsf{D}^3 + \mathsf{D}^4.$$

Correspondingly, given $\boldsymbol{u}(\mathsf{D}) = 1 + \mathsf{D}^2 + \mathsf{D}^4$ we can obtain the information sequence $\boldsymbol{u}$ from the coefficients of $\boldsymbol{u}(\mathsf{D})$ as

$$\boldsymbol{u}(\mathsf{D}) = 1 + \mathsf{D}^2 + \mathsf{D}^4 = 1 \cdot \mathsf{D}^0 + 0 \cdot \mathsf{D}^1 + 1 \cdot \mathsf{D}^2 + 0 \cdot \mathsf{D}^3 + 1 \cdot \mathsf{D}^4 \longrightarrow \boldsymbol{u} = (1, 0, 1, 0, 1, \ldots).$$

Convolution in time domain reverts to multiplication in the D-transform domain, i.e., (9.3) can be written in the D-transform domain as

$$\boldsymbol{c}_j(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_j(\mathsf{D}), \tag{9.4}$$

where the indeterminate D can be regarded as the delay operator. The delay operator D is equivalent to the discrete-time delay operator z^{-1} used in linear time-invariant sytems. However, in the coding literature the operator D is more commonly used.

Using the D-transform, we can rewrite the encoding operations in (9.1) and (9.2) in a more compact form as

$$\begin{aligned}\boldsymbol{c}(\mathsf{D}) &= (\boldsymbol{c}_1(\mathsf{D})\ \boldsymbol{c}_2(\mathsf{D}))\\ &= \boldsymbol{u}(\mathsf{D})(\boldsymbol{g}_1(\mathsf{D})\ \boldsymbol{g}_2(\mathsf{D}))\\ &= \boldsymbol{u}(\mathsf{D})\boldsymbol{G}(\mathsf{D}),\end{aligned}$$

where $\boldsymbol{G}(\mathsf{D}) = (\boldsymbol{g}_1(\mathsf{D})\ \boldsymbol{g}_2(\mathsf{D}))$ is the **generator matrix** of the code and the polynomials $\boldsymbol{g}_j(\mathsf{D})$ are referred to as the **generator polynomials** of the code. In general, the codeword $\boldsymbol{c}$Â for a $R_{\mathsf{c}} = 1/2$ convolutional encoder is formed by multiplexing the bits corresponding to the power series $\boldsymbol{c}_1(\mathsf{D})$ and $\boldsymbol{c}_2(\mathsf{D})$. This is clarified in the following example.

Example 9.2 ($R_{\mathsf{c}} = 1/2$ convolutional code) For the encoder of Figure 9.1, the generator matrix is $\boldsymbol{G}(\mathsf{D}) = (\boldsymbol{g}_1(\mathsf{D})\ \boldsymbol{g}_2(\mathsf{D}))$, with generator polynomials

$$\begin{aligned}\boldsymbol{g}_1(\mathsf{D}) &= 1 + \mathsf{D} + \mathsf{D}^2\\ \boldsymbol{g}_2(\mathsf{D}) &= 1 + \mathsf{D}^2.\end{aligned}$$

The code sequences $\boldsymbol{c}_1(\mathsf{D})$ and $\boldsymbol{c}_2(\mathsf{D}))$ are obtained as

$$\begin{aligned}\boldsymbol{c}_1(\mathsf{D}) &= \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_1(\mathsf{D})\\ \boldsymbol{c}_2(\mathsf{D}) &= \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_2(\mathsf{D}),\end{aligned}$$

from which we obtain the code sequence $\boldsymbol{c}(\mathsf{D})$ as $\boldsymbol{c}(\mathsf{D}) = (\boldsymbol{c}_1(\mathsf{D})\ \boldsymbol{c}_2(\mathsf{D}))$.

Assume that we want to encode the information sequence $\boldsymbol{u} = (1, 0, 1, 0, 0, 0, \ldots)$, and that the encoder is initialized to the all-zero state. In the D-transform domain we get $\boldsymbol{u}(\mathsf{D}) = 1 + \mathsf{D}^2$, and

$$\begin{aligned}\boldsymbol{c}_1(\mathsf{D}) &= (1 + \mathsf{D}^2)(1 + \mathsf{D} + \mathsf{D}^2) = 1 + \mathsf{D} + \mathsf{D}^3 + \mathsf{D}^4\\ \boldsymbol{c}_2(\mathsf{D}) &= (1 + \mathsf{D}^2)(1 + \mathsf{D}^2) = 1 + \mathsf{D}^4,\end{aligned}$$

which are the D-transform representation of the sequences $\boldsymbol{c}_1 = (1, 1, 0, 1, 1, \ldots)$ and $\boldsymbol{c}_2 = (1, 0, 0, 0, 1, \ldots)$, respectively. Multiplexing $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ we obtain the code sequence $\boldsymbol{c} = (1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, \ldots)$. Note that if we encode the sequence $\boldsymbol{u} = (1, 0, 1, 0, 0, 0, \ldots)$ using the finite-state machine of Figure 9.1 we obtain the same code sequence $\boldsymbol{c}$.

Let us generalize the $R_{\mathsf{c}} = 1/2$ convolutional code to any pair (n, k). An (n, k) convolutional encoder is described by a $k \times n$ generator matrix $\boldsymbol{G}(\mathsf{D})$ in the form

$$\boldsymbol{G}(\mathsf{D}) = \begin{pmatrix} \boldsymbol{g}_{11}(\mathsf{D}) & \ldots & \boldsymbol{g}_{1n}(\mathsf{D})\\ \vdots & \ddots & \vdots\\ \boldsymbol{g}_{k1}(\mathsf{D}) & \ldots & \boldsymbol{g}_{kn}(\mathsf{D}) \end{pmatrix}.$$

Example 9.3 ($R_{\mathsf{c}} = 2/3$ convolutional code) Consider the $R_{\mathsf{c}} = 2/3$ convolutional encoder given by the generator matrix

$$\boldsymbol{G}(\mathsf{D}) = \begin{pmatrix} 1 & 0 & 0\\ 0 & 1 + \mathsf{D}^2 & \mathsf{D} \end{pmatrix} \tag{9.5}$$

with generator polynomials

$$\begin{aligned}&\boldsymbol{g}_{11}(\mathsf{D}) = 1, && \boldsymbol{g}_{12}(\mathsf{D}) = 0, && \boldsymbol{g}_{13}(\mathsf{D}) = 0,\\ &\boldsymbol{g}_{21}(\mathsf{D}) = 0, && \boldsymbol{g}_{21}(\mathsf{D}) = 1 + \mathsf{D}^2, && \boldsymbol{g}_{23}(\mathsf{D}) = \mathsf{D}.\end{aligned}$$

The code sequence $\boldsymbol{c}(\mathsf{D}) = (\boldsymbol{c}_1(\mathsf{D})\ \boldsymbol{c}_2(\mathsf{D})\ \boldsymbol{c}_3(\mathsf{D}))$ is obtained from the information sequence

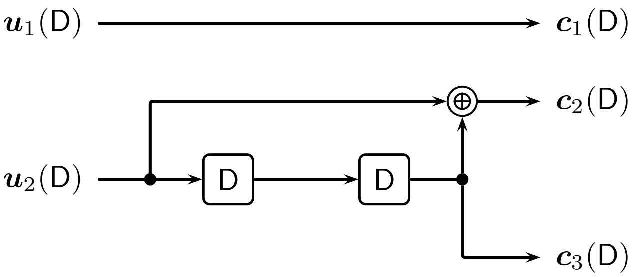

$\boldsymbol{u}_1(\mathsf{D})$
$\boldsymbol{c}_1(\mathsf{D})$
$\boldsymbol{c}_2(\mathsf{D})$
$\boldsymbol{u}_2(\mathsf{D})$
D
D
$\boldsymbol{c}_3(\mathsf{D})$

Figure 9.2: Block diagram of the convolutional encoder with parameters $(n, k) = (3, 2)$ of Example 9.3 with generator matrix given in (9.5).

$\boldsymbol{u}(\mathsf{D}) = (\boldsymbol{u}_1(\mathsf{D})\ \boldsymbol{u}_2(\mathsf{D}))$ as

$$\begin{aligned}\boldsymbol{c}_1(\mathsf{D}) &= \boldsymbol{u}_1(\mathsf{D})\boldsymbol{g}_{11}(\mathsf{D}) + \boldsymbol{u}_2(\mathsf{D})\boldsymbol{g}_{21}(\mathsf{D})\\ \boldsymbol{c}_2(\mathsf{D}) &= \boldsymbol{u}_1(\mathsf{D})\boldsymbol{g}_{12}(\mathsf{D}) + \boldsymbol{u}_2(\mathsf{D})\boldsymbol{g}_{22}(\mathsf{D})\\ \boldsymbol{c}_3(\mathsf{D}) &= \boldsymbol{u}_1(\mathsf{D})\boldsymbol{g}_{13}(\mathsf{D}) + \boldsymbol{u}_2(\mathsf{D})\boldsymbol{g}_{23}(\mathsf{D}).\end{aligned}$$

The block diagram of the encoder is depicted in Figure 9.2.

Analogous to linear block codes, the generator matrix defines completely the encoder, i.e., the mapping between information words and codewords. Therefore, we use *encoder* and *generator matrix* interchangeably, because they both describe the same mapping.

An encoder that has only polynomial entries in its generator matrix is said to be a **feedforward** encoder (this is the case of the encoders of Figures 9.1 and 9.2), while an encoder that has rational functions in its generator matrix is said to be a **recursive** encoder.

9.1.2 Systematic Encoders

Similar to linear block codes, for each generator matrix $\boldsymbol{G}(\mathsf{D})$ it is possible to obtain a systematic generator matrix $\boldsymbol{G}_{\mathsf{s}}(\mathsf{D})$ in the form $\boldsymbol{G}_{\mathsf{s}}(\mathsf{D}) = (\boldsymbol{I}_k\ \ \boldsymbol{P}(\mathsf{D}))$ by linear combinations of the rows of $\boldsymbol{G}(\mathsf{D})$, combined with possible column permutations. In particular, each feedforward convolutional encoder has an equivalent recursive, systematic encoder. For rate $R_{\mathsf{c}} = 1/2$, an equivalent systematic encoder is easily obtained by dividing all polynomials of the generator matrix by one of the polynomials.

Example 9.4 ($R_{\mathsf{c}} = 1/2$ code with recursive, systematic encoder) We can obtain a recursive systematic encoder from the feedforward encoder of Example 9.2 (Figure 9.1) defined by the generator matrix

$$\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2\ \ 1 + \mathsf{D}^2) \tag{9.6}$$

by dividing the generator polynomials of $\boldsymbol{G}(\mathsf{D})$ by $1 + \mathsf{D} + \mathsf{D}^2$, hence obtaining

$$\boldsymbol{G}_{\mathsf{s}}(\mathsf{D}) = \frac{\boldsymbol{G}(\mathsf{D})}{1 + \mathsf{D} + \mathsf{D}^2} = \left(1\ \ \frac{1 + \mathsf{D}^2}{1 + \mathsf{D} + \mathsf{D}^2}\right). \tag{9.7}$$

The encoder defined by (9.7) is drawn in Figure 9.3. Note that the encoder is recursive. Notice also that $\boldsymbol{G}(\mathsf{D})$ in (9.6) and $\boldsymbol{G}_{\mathsf{s}}(\mathsf{D})$ in (9.7) generate the same code, i.e., the same list of codewords. However, $\boldsymbol{G}(\mathsf{D})$

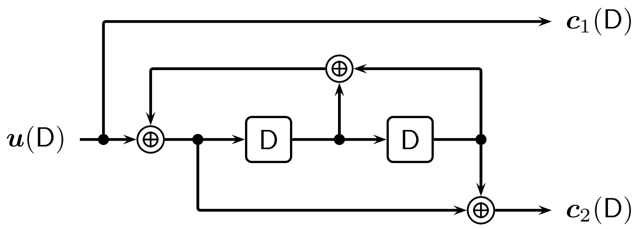

$\boldsymbol{c}_1(\mathsf{D})$
$\boldsymbol{u}(\mathsf{D})$
D
D
$\boldsymbol{c}_2(\mathsf{D})$

Figure 9.3: Recursive systematic convolutional encoder with generator matrix $\boldsymbol{G}_\mathsf{s}(\mathsf{D}) = \left(1 \;\; \frac{1+\mathsf{D}^2}{1+\mathsf{D}+\mathsf{D}^2}\right)$, equivalent to the convolutional encoder of Figure 9.1.

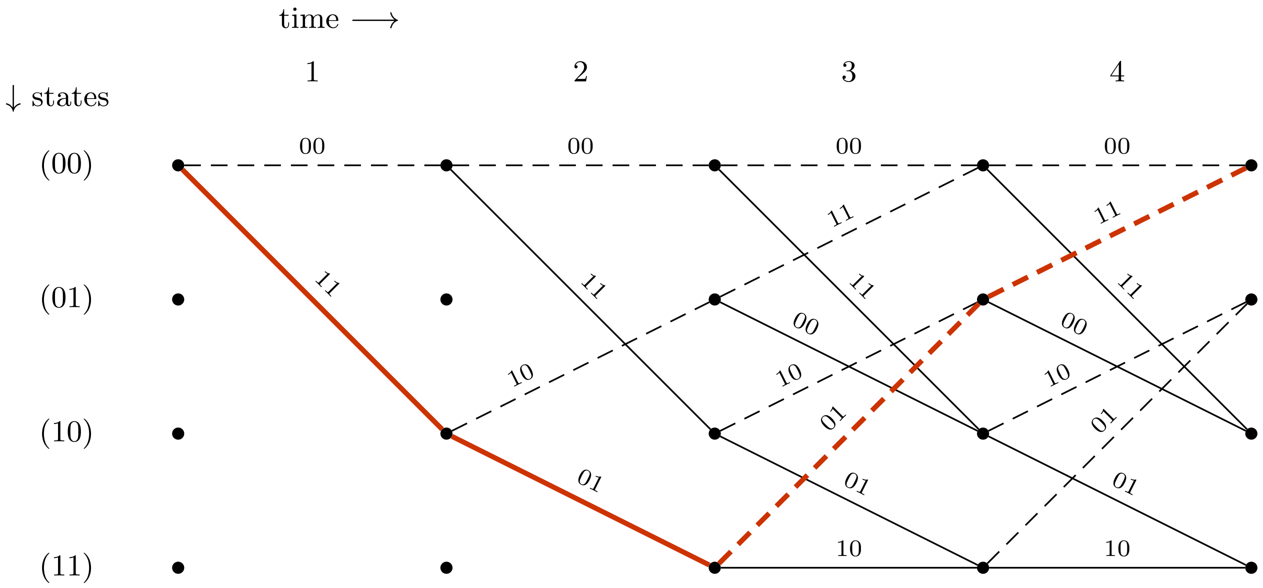


Figure 9.4: Trellis representation of a convolutional code with constraint length $\nu = 2$. Dashed edges correspond to input $u_i = 0$, and solid edges to $u_i = 1$. The edges are labeled with the two coded output bits. The red lines correspond to the path through the trellis resulting from the encoding of $\boldsymbol{u} = (1, 1, 0, 0)$. From the trellis we can read the output bits: $\boldsymbol{c}_1 = (1, 0, 0, 1)$ and $\boldsymbol{c}_2 = (1, 1, 1, 1)$.

and $\boldsymbol{G}_\mathsf{s}(\mathsf{D})$ correspond to different mappings between information words and codewords, i.e., to different encoders.

9.2 Graphical Representation of Convolutional Codes: The Trellis Diagram

In this section, we introduce a graphical representation of convolutional codes, called the **trellis diagram**, that is very useful for code analysis. Furthermore, the Viterbi algorithm for ML decoding of convolutional codes is based on the trellis diagram. The trellis diagram is obtained by replicating all possible states of the encoder at each time instant i and captures how the state of the encoder evolves over time. As an example, we will focus again on the rate $R_\mathsf{c} = 1/2$, memory $\nu = 2$ (i.e., 4 states) convolutional encoder with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \quad 1 + \mathsf{D}^2)$ of Figure 9.1. A part of the corresponding trellis (for time instants $i = 1, 2, 3$, and 4) is shown in Figure 9.4. In the trellis, the four nodes on the same vertical represent the four possible states $(0, 0), (0, 1), (1, 0)$, and $(1, 1)$. Horizontally, we represent time i, which we will sometimes refer to as the trellis depth. We assume that the encoder is initialized to the all-zero state, i.e., $\boldsymbol{s}_1 = (0, 0)$.

At each time instant i, the convolutional encoder encodes one input bit u_i, generating two output (code) bits $c_{1,i}$ and $c_{2,i}$, and moving the state of the encoder to state $\boldsymbol{s}_{i+1}$. In the figure, we use the convention that a dashed edge corresponds to input bit $u_i = 0$ and a solid edge to input bit $u_i = 1$. Since the initial state of the encoder at time $i = 1$ is $\boldsymbol{s}_1 = (0, 0)$, at time instant $i = 2$ the state of the encoder can take on two possible values: it is $\boldsymbol{s}_2 = (0, 0)$ if the input bit is $u_1 = 0$, as indicated by the dashed edge, or $\boldsymbol{s}_2 = (1, 0)$ if the input bit $u_1 = 1$, as indicated by the solid edge. The edges are labeled with the corresponding code bits $c_{1,i}$ and $c_{2,i}$. Starting from state $\boldsymbol{s}_1 = (0, 0)$, an input bit $u_1 = 1$ will produce the code bits $c_{1,1} = 1$ and $c_{2,1} = 1$, and bring the encoder to state $\boldsymbol{s}_2 = (1, 0)$.

We remark that any code sequence corresponds to a path through the trellis, where the code bits label the edges of the path. For example, the orange line in Figure 9.4 shows the evolution of the encoder for the input sequence $\boldsymbol{u} = (1, 1, 0, 0, \ldots)$. From the trellis diagram, we can immediately see that this information sequence produces the sequence of states $\boldsymbol{s}_1 = (0, 0)$, $\boldsymbol{s}_2 = (1, 0)$, $\boldsymbol{s}_3 = (1, 1)$, $\boldsymbol{s}_4 = (0, 1)$, and $\boldsymbol{s}_5 = (0, 0)$, and it generates the code sequences $\boldsymbol{c}_1 = (1, 0, 0, 1, \ldots)$ and $\boldsymbol{c}_2 = (1, 1, 1, 1, \ldots)$.

Notice that the structure of the trellis of a convolutional code is **time-invariant**, except at the beginning (see Figure 9.4) and at the end of the trellis (due to the trellis termination, as we will see in Section 9.3). Therefore, a convolutional code (as well as the encoder) is completely described by a single section of the trellis. The trellis is just the concatenation of several trellis sections.

9.3 Trellis Termination

While convolutional codes are stream-oriented, most practical applications require block-oriented transmission, i.e., we are interested in encoding an information word $\boldsymbol{u} = (u_1, \ldots, u_K)$ of finite length K bits into a finite-length codeword $\boldsymbol{c} = (c_1, \ldots, c_N)$ of length N bits. In other words, we would like to *transform* the (n, k) convolutional code into a block code of parameters (N, K). The process of transforming a convolutional code into a block code is usually referred to as **code termination** or **trellis termination**.

The most straightforward way to transform an (n, k) convolutional code into an (N, K) block code is to run the encoder K/k steps (or equivalently do K/k steps in the trellis[2]) and output the resulting codeword, of length $N = \frac{K}{k} n$ bits. The resulting code rate is

$$\boxed{R_\mathsf{c} = \frac{K}{N} = \frac{K}{(K/k)n} = \frac{k}{n},}$$

i.e., the same as that of the original (n, k) convolutional code. The main drawback of this trellis termination, referred to as **trellis truncation**, is that the last information bits are less reliable, since they are protected by fewer code bits. To circumvent this problem, in the following we present another trellis termination procedure that is widely used in practice.

9.3.1 Zero Termination

To overcome the problem that the last information bits are less protected, it is common to **terminate** the trellis to a known state. Typically, the trellis is terminated to the all-zero state (since the initial state is the all-zero state), hence it is referred to as **zero termination**. Zero termination requires appending νk **dummy** bits to the information sequence, whose sole purpose is to bring the state of the encoder back to the all-zero state. Obviously, adding dummy bits results in an increase in redundancy, or equivalently, a decrease

[2] We recall that at each time instant i the convolutional encoder encodes k information bits into n code bits.

in code rate. The main principle is as follows: We run the encoder K/k steps (K/k steps in the trellis). This generates $N = \frac{K}{k}n$ code bits. We then run the encoder ν additional steps by appending νk dummy bits to the information sequence to terminate the encoder to the all-zero state, thus generating νn additional code bits. The resulting block code is a code of length

$$N = \frac{K}{k}n + \nu n.$$

Thus, the code rate of the terminated convolutional code is

$$R_\mathsf{c} = \frac{K}{N} = \frac{K}{(K/k)n + \nu n} = \frac{k}{n}\frac{1}{1 + k\nu/K} < \frac{k}{n}. \tag{9.8}$$

We observe from (9.8) that zero termination implies a rate loss: The rate R_c is decreased with respect to the rate of the underlying (n, k) convolutional code.[3] However, note that when K grows very large, the term $k\nu/K$ goes to zero and hence the rate loss is negligible,

$$R_\mathsf{c} = \frac{K}{N} = \frac{k}{n}\frac{1}{1 + k\nu/K} \xrightarrow{K\to\infty} \frac{k}{n}.$$

For rate-$1/n$ feedforward encoders, zero-termination is straightforward: We append ν zero bits to the information sequence, thus moving the state of the encoder to the all-zero state. For recursive encoders, zero-termination is a bit more involved and requires solving a system of equations.

Remark 9.1 In practice, convolutional codes are never truncated. Both zero termination and tailbiting termination are widely used for both stand-alone convolutional codes and when they are considered as component codes in a turbo-like code.

9.4 Computation of the Minimum Hamming Distance

As for block codes, the error correction and error detection capabilities of convolutional codes are directly related to the distance properties of the encoded sequences and, in particular, to the minimum Hamming distance (for convolutional codes, the minimum Hamming distance is sometimes referred to as **free distance**). As every code sequence of a convolutional code corresponds to a path in the trellis, the minimum Hamming distance can be computed using the trellis diagram.

Since convolutional codes are linear codes, we can assume that the all-zero codeword is transmitted. Then the minimum Hamming distance can be obtained from the trellis by looking at all sequences (paths in the trellis) which diverge from the all-zero state at time instant $i = 1$ and merge into it later. The minimum Hamming distance corresponds to the weight of the path with smallest weight that starts at the all-zero state at time $i = 1$, deviates from it, and later remerges to the all-zero state. For the convolutional code of Figure 9.4, it is easy to see that the minimum Hamming distance distance is 5, and corresponds to the input sequence $\boldsymbol{u} = (1, 0, 0, \dots, 0)$, i.e., the information sequence with a single one in the first position.

We can use the following algorithm to compute the minimum Hamming distance d_min. Denote by d_temp a temporary distance. Also, denote by $d_\mathsf{c}(i)$ the minimum Hamming weight of all trellis paths up to trellis depth $i > 1$ that have diverged from the all-zero state at $i = 1$. Then,

1. Set the trellis depth $i = 1$ and $d_\mathsf{temp} = \infty$.

[3] A clever way to circumvent the rate loss induced by zero termination while still maintaining the advantage of trellis termination is via **tailbiting**. However, we will not cover this trellis termination technique in this course.

2. Set $i \to i+1$.

3. Compute $d_{\mathsf{c}}(i)$.

4. If a sequence of weight $d < d_{\mathsf{temp}}$ at trellis depth i merges to the all-zero state fix $d_{\mathsf{temp}} = d$.

5. If $d_{\mathsf{c}}(i) > d_{\mathsf{temp}}$ go to Step 7.

6. Return to Step 2.

7. Set $d_{\mathsf{min}} = d_{\mathsf{temp}}$ and stop.

9.5 Maximum Likelihood Decoding of Convolutional Codes

In this section, we discuss ML decoding of convolutional codes. We focus again on the rate $R_{\mathsf{c}} = 1/2$ convolutional code with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1+\mathsf{D}+\mathsf{D}^2 \quad 1+\mathsf{D}^2)$ of Figure 9.1 whose first trellis sections are depicted in Figure 9.4. We will address first hard-decision decoding and then soft-decision decoding assuming an AWGN channel and BPSK signaling.

9.5.1 Hard-Decision Decoding

As discussed in Section 7.4.1, for hard-decision decoding the channel can be modeled as a BSC. The received sequence is

$$\bar{\boldsymbol{y}} = (\bar{\boldsymbol{y}}_1 \, \bar{\boldsymbol{y}}_2),$$

where

$$\begin{aligned}\bar{\boldsymbol{y}}_1 &= \boldsymbol{c}_1 + \boldsymbol{e}_1\\ \bar{\boldsymbol{y}}_2 &= \boldsymbol{c}_2 + \boldsymbol{e}_2,\end{aligned}$$

with $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ being the error patterns introduced by the channel.

The ML decoder selects the codeword $\boldsymbol{c}$ that is closest to $\bar{\boldsymbol{y}}$ in terms of Hamming distance,

$$\boxed{\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c}, \bar{\boldsymbol{y}}).} \tag{9.9}$$

We assume block-oriented transmission, i.e., the convolutional code is terminated. Let K be the information block length. If the encoder is terminated to the all-zero state, we run the encoder $L = K/k + \nu$ times (which correspond to L steps in the trellis). $d_{\mathsf{H}}(\boldsymbol{c}, \bar{\boldsymbol{y}})$ can then be written as

number of steps
for Viterbi

$$L = \frac{\text{Length of information block}}{\text{number of dimintion.}} + \text{number of memory}$$

$$\begin{aligned}d_{\mathsf{H}}(\boldsymbol{c}, \bar{\boldsymbol{y}}) &= \sum_{i=1}^{L} \left(d_{\mathsf{H}}(c_{1,i}, \bar{y}_{1,i}) + d_{\mathsf{H}}(c_{2,i}, \bar{y}_{2,i})\right)\\ &= \sum_{i=1}^{L} d_{\mathsf{H}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i)\\ &= \sum_{i=1}^{L} \lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i),\end{aligned} \tag{9.10}$$

where we have defined $\boldsymbol{c}^i = (c_{1,i}, c_{2,i})$, $\bar{\boldsymbol{y}}^i = (\bar{y}_{1,i}, \bar{y}_{2,i})$ and

$$\lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i) \triangleq d_{\mathsf{H}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i).$$

Therefore, using (9.10) the ML decoder rule for hard-decision decoding (9.9) can be rewritten as

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i). \tag{9.11}$$

9.5.2 Soft-Decision Decoding

For transmission over the AWGN channel with BPSK signaling, the received vector is

$$\boldsymbol{y} = (\boldsymbol{y}_1\,\boldsymbol{y}_2),$$

where

$$\begin{aligned}\boldsymbol{y}_1 &= (-1)^{\boldsymbol{c}_1} + \boldsymbol{n}_1 = \boldsymbol{x}_1 + \boldsymbol{n}_1\\ \boldsymbol{y}_2 &= (-1)^{\boldsymbol{c}_2} + \boldsymbol{n}_2 = \boldsymbol{x}_2 + \boldsymbol{n}_2,\end{aligned}$$

where $(-1)^{\boldsymbol{c}_1} = ((-1)^{c_{1,1}}, (-1)^{c_{1,2}}, \dots) = (x_{1,1}, x_{1,2}, \dots) = \boldsymbol{x}_1$ and $(-1)^{\boldsymbol{c}_2} = ((-1)^{c_{2,1}}, (-1)^{c_{2,2}}, \dots) = (x_{2,1}, x_{2,2}, \dots) = \boldsymbol{x}_2$ are the BPSK-modulated sequences, and $\boldsymbol{n}_1 = (n_{1,1}, n_{1,2}, \dots)$ and $\boldsymbol{n}_2 = (n_{2,1}, n_{2,2}, \dots)$, with $n_{1,i}$ and $n_{2,i}$ being realizations of a Gaussian random variable with zero mean and variance $\sigma^2 = \mathsf{N}_0/2$.

The ML decoder finds the codeword $\boldsymbol{c}$ that is closest to $\boldsymbol{y}$ in terms of Euclidean distance (see Section 7.4.3 and (7.9)),

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{E}}^2(\boldsymbol{x}, \boldsymbol{y}), \tag{9.12}$$

where we considered the squared Euclidean distance for convenience. In fact, we can rewrite $d_{\mathsf{E}}^2(\boldsymbol{x}, \boldsymbol{y})$ as

$$\begin{aligned} d_{\mathsf{E}}^2(\boldsymbol{c}, \boldsymbol{y}) &= \sum_{i=1}^{L} \left\| \boldsymbol{y}^i - (-1)^{\boldsymbol{c}^i} \right\|^2 \\ &= \sum_{i=1}^{L} \left(|y_{1,i} - (-1)^{c_{1,i}}|^2 + |y_{2,i} - (-1)^{c_{2,i}}|^2 \right) \\ d_E^2(c,y) &= \sum_{i=1}^{L} \lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i, \boldsymbol{y}^i), \end{aligned} \tag{9.13}$$

where we have defined $\boldsymbol{y}^i \triangleq (y_{1,i}, y_{2,i})$, $\boldsymbol{c}^i \triangleq (c_{1,i}, c_{2,i})$, and $\lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i, \boldsymbol{y}^i) \triangleq \left\| \boldsymbol{y}^i - (-1)^{\boldsymbol{c}^i} \right\|^2$.

Therefore, using (9.13) the ML decoder rule for soft-decision decoding (9.12) can be rewritten as

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i, \boldsymbol{y}^i). \tag{9.14}$$

For soft-decision decoding, it may be more convenient to consider the correlation metric (7.10),

$$\begin{aligned} \hat{\boldsymbol{c}} &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} (y_{1,i}(-1)^{c_{1,i}} + y_{2,i}(-1)^{c_{2,i}}) \\ &= \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i^{\mathsf{SOFT-corr}}(\boldsymbol{c}^i, \boldsymbol{y}^i) \end{aligned} \tag{9.15}$$

We see from (9.11) and (9.14) that both ML hard-decision decoding and soft-decision decoding correspond to solving a problem of the form

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i. \tag{9.16}$$

where $\lambda_i = \lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i)$ for hard-decision decoding and $\lambda_i = \lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i, \boldsymbol{y}^i)$ for soft-decision decoding. Alternatively, for soft-decision decoding

$$\hat{\boldsymbol{c}} = \arg\max_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i. \tag{9.17}$$

where $\lambda_i = \lambda_i^{\mathsf{SOFT-corr}}(\boldsymbol{c}^i, \boldsymbol{y}^i)$.

Since every codeword $\boldsymbol{c}$ corresponds to a path in the trellis, we can solve (9.16) and (9.17) using the trellis diagram. We will refer to the quantities λ_i as the **branch metrics** for trellis section i. Clearly, the number of computations required to solving (9.16) and (9.17) using brute force is enormous, since there are 2^K codewords. Therefore, a brute force approach is unfeasible. However, there exists an algorithm to solve (9.16) and (9.17) with a complexity that grows linearly with the information block length K. This algorithm is the famous **Viterbi algorithm** and is discussed in the next section.

9.6 The Viterbi Algorithm

Andrew Viterbi noticed that the complexity of computing (9.16) and (9.17) can be reduced to be linear with K with no loss of optimality under the following observation. Denote by Γ_ℓ the accumulated metric between a codeword $\boldsymbol{c}$ and the received vector $\boldsymbol{y}$ (or $\bar{\boldsymbol{y}}$) up to time ℓ, i.e.,

$$\boxed{\Gamma_\ell(\boldsymbol{c}) = \sum_{i=1}^{\ell} \lambda_i.} \tag{9.18}$$

For simplicity, we will consider Viterbi to solve the minimization problem (9.16) for both hard-decision decoding and soft-decision decoding and later indicate the (very minor) differences required to solve the maximization problem (9.17) when using the correlation metric for soft-decision decoding.

Consider two code sequences $\boldsymbol{c}$ and $\tilde{\boldsymbol{c}}$ (corresponding to two paths through the trellis) that diverge from the all-zero state at time $i = 1$, remerge to the same state $\boldsymbol{s}_\ell$ at time $\ell > i$, and are equal for all $t > \ell$. The significance of this remerging is that if the path corresponding to $\boldsymbol{c}$ is such that $\Gamma_\ell(\boldsymbol{c}) > \Gamma_\ell(\tilde{\boldsymbol{c}})$ at trellis depth ℓ, then it follows $\Gamma_L(\boldsymbol{c}) > \Gamma_L(\tilde{\boldsymbol{c}})$ at trellis depth L (at the end of the trellis). Consequently, we can safely discard $\boldsymbol{c}$. The path that is maintained is called the **survivor**. This argument applies to all merging paths in the trellis, and it is the main principle of the **Viterbi algorithm**.

Let us introduce a few definitions.

1. $\lambda_i(\boldsymbol{s}', \boldsymbol{s})$ is the branch metric from state $\boldsymbol{s}'$ at time i to state $\boldsymbol{s}$ at time $i+1$, i.e., $\lambda_i(\boldsymbol{s}', \boldsymbol{s}) = \lambda_i(\boldsymbol{c}^i, \boldsymbol{y}^i)$ (or $\lambda_i(\boldsymbol{s}', \boldsymbol{s}) = \lambda_i(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i)$), where with some abuse of language we denote by $\boldsymbol{c}^i$ the code bits of the branch $\boldsymbol{s}' \to \boldsymbol{s}$, and $\boldsymbol{y}^i$ (or $\bar{\boldsymbol{y}}^i$ for hard-decision decoding) are the received symbols for trellis section i.

2. $\Gamma_i(\boldsymbol{s})$ is the cumulative metric for the survivor state $\boldsymbol{s}$ at time i, i.e., it is the sum of the metrics for the surviving path.

3. $\Gamma_{i+1}(\boldsymbol{s}', \boldsymbol{s})$ is the *tentative* cumulative metric for the path from $\boldsymbol{s}'$ at time i to $\boldsymbol{s}$ at time $i+1$, i.e., $\Gamma_{i+1}(\boldsymbol{s}', \boldsymbol{s}) = \Gamma_i(\boldsymbol{s}') + \lambda_i(\boldsymbol{s}', \boldsymbol{s})$.

The Viterbi algorithm is given in Algorithm 1 below.

For soft-decision decoding using (9.17), the same algorithm can be used with the only modification of using max instead of min in Step 5. In the following we give a complete example of Viterbi decoding for hard-decision decoding.

Algorithm 1 Viterbi Algorithm

```
Initialization. Set $\Gamma_1(00) = 0$ and $\Gamma_1(\boldsymbol{s}) = \infty$ for all $\boldsymbol{s} \in \{0,1\}^\nu$. (The encoder, and hence the trellis, is initialized to the all-zero state).
for $i = 2$ to $L$ do (Add–Compare–Select iteration)
    Compute the possible branch metrics $\lambda_{i-1}(\boldsymbol{s}', \boldsymbol{s})$.
    For each state $\boldsymbol{s}'$ at time $i-1$ and all possible states $\boldsymbol{s}$ at time $i$ that can be reached from $\boldsymbol{s}'$, compute the metrics $\Gamma_i(\boldsymbol{s}', \boldsymbol{s}) = \Gamma_{i-1}(\boldsymbol{s}') + \lambda_{i-1}(\boldsymbol{s}', \boldsymbol{s})$ for the paths extending from $\boldsymbol{s}'$ to $\boldsymbol{s}$.
    For each state $\boldsymbol{s}$ at time $i$, select and store the path possessing the minimum among the metrics $\Gamma_i(\boldsymbol{s}', \boldsymbol{s})$. The cumulative metric for state $\boldsymbol{s}$ will be $\Gamma_i(\boldsymbol{s}) = \min_{\boldsymbol{s}'} \Gamma_i(\boldsymbol{s}', \boldsymbol{s})$.
end for
Decision. Since we assume the termination of the trellis to the all-zero state, after the final add-compare-select iteration, the ML trellis path (i.e., the ML codeword) will be the survivor at the all-zero state.
```

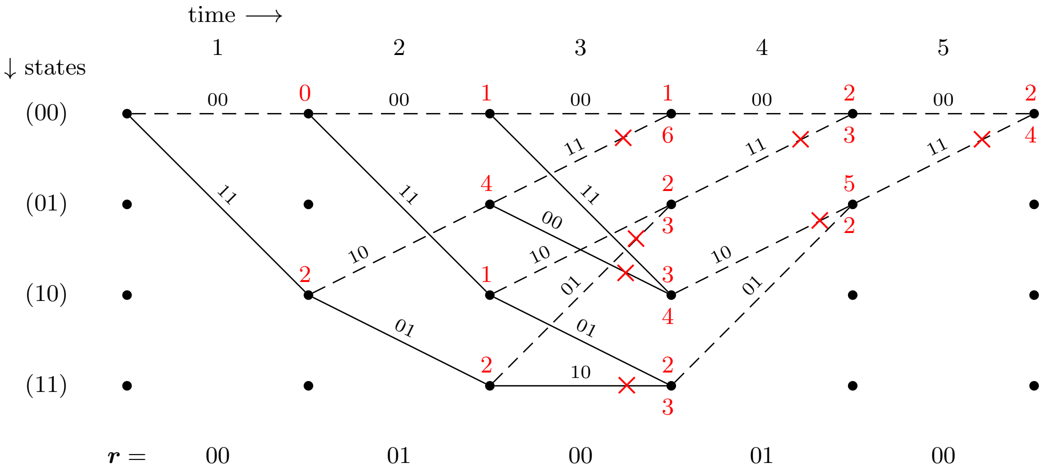


Figure 9.5: Decoding of the received vector $\bar{\boldsymbol{y}} = (0,0,0,1,0,0,0,1,0,0)$ using the Viterbi algorithm. The encoder is $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \;\; 1 + \mathsf{D}^2)$.

Example 9.5 (Hard-decision Viterbi decoding of the $R_\mathsf{c} = 1/2$ convolutional code) We consider again the convolutional encoder with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \;\; 1 + \mathsf{D}^2)$ and hard-decision decoding (i.e., transmission over the BSC). Assume that the information block length is $K = 3$, and the encoder is zero-terminated. Therefore, we run the encoder $L = K/k + \nu = 3 + 2 = 5$ steps. Also, we assume that the encoder is initialized to the all-zero state before starting encoding.

Suppose that we receive $\bar{\boldsymbol{y}} = (0,0,0,1,0,0,0,1,0,0)$. We would like to find the ML codeword using the Viterbi algorithm. We describe all the steps in detail in the following and show the Viterbi decoding graphically in Figure 9.5.

1. Set $\Gamma_1(00) = 0$ and $\Gamma_1(\boldsymbol{s}') = \infty$ for $\boldsymbol{s}' \in \{(0,1),(1,0),(1,1)\}$.

2. For $i = 2$ we compute

$$\lambda_1(00,00) = d_\mathsf{H}(00,00) = 0$$
$$\lambda_1(00,10) = d_\mathsf{H}(11,00) = 2,$$

and

$$\begin{aligned}
\Gamma_2(00,00) &= \Gamma_1(00) + \lambda_1(00,00) = 0 + 0 = 0\\
\Gamma_2(00,10) &= \Gamma_1(00) + \lambda_1(00,10) = 0 + 2 = 2.
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
\Gamma_2(00) &= 0\\
\Gamma_2(10) &= 2.
\end{aligned}$$

3. For $i = 3$ we compute

$$\begin{aligned}
\lambda_2(00,00) &= d_{\mathsf{H}}(00,01) = 1\\
\lambda_2(10,01) &= d_{\mathsf{H}}(10,01) = 2\\
\lambda_2(00,10) &= d_{\mathsf{H}}(11,01) = 1\\
\lambda_2(10,11) &= d_{\mathsf{H}}(01,01) = 0,
\end{aligned}$$

   and

$$\begin{aligned}
\Gamma_3(00,00) &= \Gamma_2(00) + \lambda_2(00,00) = 0 + 1 = 1\\
\Gamma_3(10,01) &= \Gamma_2(10) + \lambda_2(10,01) = 2 + 2 = 4\\
\Gamma_3(00,10) &= \Gamma_2(00) + \lambda_2(00,10) = 0 + 1 = 1\\
\Gamma_3(10,11) &= \Gamma_2(10) + \lambda_2(10,11) = 2 + 0 = 2.
\end{aligned}$$

   Therefore, we obtain

$$\begin{aligned}
\Gamma_3(00) &= 1\\
\Gamma_3(01) &= 4\\
\Gamma_3(10) &= 1\\
\Gamma_3(11) &= 2.
\end{aligned}$$

4. For $i = 4$ we compute

$$\begin{aligned}
\lambda_3(00,00) &= d_{\mathsf{H}}(00,00) = 0\\
\lambda_3(00,10) &= d_{\mathsf{H}}(11,00) = 2\\
\lambda_3(01,00) &= d_{\mathsf{H}}(11,00) = 2\\
\lambda_3(01,10) &= d_{\mathsf{H}}(00,00) = 0\\
\lambda_3(10,01) &= d_{\mathsf{H}}(10,00) = 1\\
\lambda_3(10,11) &= d_{\mathsf{H}}(01,00) = 1\\
\lambda_3(11,01) &= d_{\mathsf{H}}(01,00) = 1\\
\lambda_3(11,11) &= d_{\mathsf{H}}(10,00) = 1,
\end{aligned}$$

and

$$\begin{aligned}
\Gamma_4(00,00) &= \Gamma_3(00) + \lambda_3(00,00) = 1 + 0 = 1\\
\Gamma_4(00,10) &= \Gamma_3(00) + \lambda_3(00,10) = 1 + 2 = 3\\
\Gamma_4(01,00) &= \Gamma_3(01) + \lambda_3(01,00) = 4 + 2 = 6\\
\Gamma_4(01,10) &= \Gamma_3(01) + \lambda_3(01,10) = 4 + 0 = 4\\
\Gamma_4(10,01) &= \Gamma_3(10) + \lambda_3(10,01) = 1 + 1 = 2\\
\Gamma_4(10,11) &= \Gamma_3(10) + \lambda_3(10,11) = 1 + 1 = 2\\
\Gamma_4(11,01) &= \Gamma_3(11) + \lambda_3(11,01) = 2 + 1 = 3\\
\Gamma_4(11,11) &= \Gamma_3(11) + \lambda_3(11,11) = 2 + 1 = 3.
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
\Gamma_4(00) &= \min\{\Gamma_4(00,00), \Gamma_4(01,00)\} = \Gamma_4(00,00) = 1\\
\Gamma_4(01) &= \min\{\Gamma_4(10,01), \Gamma_4(11,01)\} = \Gamma_4(10,01) = 2\\
\Gamma_4(10) &= \min\{\Gamma_4(00,10), \Gamma_4(01,10)\} = \Gamma_4(00,10) = 3\\
\Gamma_4(11) &= \min\{\Gamma_4(10,11), \Gamma_4(11,11)\} = \Gamma_4(10,11) = 2.
\end{aligned}$$

5. For $i = 5$ we compute

$$\begin{aligned}
\lambda_4(00,00) &= d_{\mathsf{H}}(00,01) = 1\\
\lambda_4(01,00) &= d_{\mathsf{H}}(11,01) = 1\\
\lambda_4(10,01) &= d_{\mathsf{H}}(10,01) = 2\\
\lambda_4(11,01) &= d_{\mathsf{H}}(01,01) = 0,
\end{aligned}$$

and

$$\begin{aligned}
\Gamma_5(00,00) &= \Gamma_4(00) + \lambda_4(00,00) = 1 + 1 = 2\\
\Gamma_5(01,00) &= \Gamma_4(01) + \lambda_4(01,00) = 2 + 1 = 3\\
\Gamma_5(10,01) &= \Gamma_4(10) + \lambda_4(10,01) = 3 + 2 = 5\\
\Gamma_5(11,01) &= \Gamma_4(11) + \lambda_4(11,01) = 2 + 0 = 2.
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
\Gamma_5(00) &= \min\{\Gamma_5(00,00), \Gamma_5(01,00)\} = \Gamma_5(00,00) = 2\\
\Gamma_5(01) &= \min\{\Gamma_5(10,01), \Gamma_5(11,01)\} = \Gamma_5(11,01) = 2.
\end{aligned}$$

6. Finally, for $i = 6$ we compute

$$\begin{aligned}
\lambda_5(00,00) &= d_{\mathsf{H}}(00,00) = 0\\
\lambda_5(01,00) &= d_{\mathsf{H}}(11,00) = 2,
\end{aligned}$$

and

$$\Gamma_6(00,00) = \Gamma_5(00) + \lambda_5(00,00) = 2 + 0 = 2$$
$$\Gamma_6(01,00) = \Gamma_5(01) + \lambda_5(01,00) = 2 + 2 = 4.$$

Therefore, we obtain

$$\Gamma_6(00) = \min\{\Gamma_6(00,00), \Gamma_6(01,00)\} = \Gamma_6(00,00) = 2.$$

At the end of the trellis we have a single survivor with accumulated distance 2: The ML codeword is the codeword corresponding to the path across the states $(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0)$, i.e., the all-zero codeword, and we decode $\bar{\boldsymbol{y}} = (0,0,0,1,0,0,0,1,0,0)$ onto $\hat{\boldsymbol{c}} = (0,0,0,0,0,0,0,0,0,0)$.

9.7 Bounds on the Probability of Error

To gain insight into the performance of a code and avoid running time-consuming simulations, it is desirable to derive analytical expressions of the error probability in terms of the power efficiency $\mathsf{E_b/N_0}$ and the code parameters. The performance of convolutional codes and linear codes in general depends on the minimum Hamming distance and, more generally, the weight enumerator (also called distance spectrum) of the code. Unfortunately, deriving the exact probability of error for long codes is unfeasible. However, as for the case of uncoded transmission with linear modulations, we can derive bounds on the error probability using the union bound. In this section, we derive bounds on the word error probability of linear block codes, denoted by $\boxed{P_{\mathsf{w}}(e)}$, i.e., the probability that a codeword $\boldsymbol{c}$ is decoded in error. The analysis is general and does not apply only to convolutional code, but to any linear code.

Let $\boldsymbol{c}$ be the transmitted codeword and $\hat{\boldsymbol{c}}$ the decoded codeword. Also, let $\Pr(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{c})$ the probability that the decoder selects a codeword $\hat{\boldsymbol{c}}$ different from $\boldsymbol{c}$ if $\boldsymbol{c}$ was transmitted and let $\Pr(\boldsymbol{c})$ the probability that codeword $\boldsymbol{c}$ is transmitted. The word error probability is given by

$$\begin{aligned} P_{\mathsf{w}} &= \sum_{\boldsymbol{c} \in \mathcal{C}} \Pr(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{c}) \Pr(\boldsymbol{c}) \\ &= \frac{1}{2^K} \sum_{\boldsymbol{c} \in \mathcal{C}} \Pr(\boldsymbol{c} \neq \hat{\boldsymbol{c}}|\boldsymbol{c}), \end{aligned}$$

where in the last equality we assumed that all codewords are equiprobable, which is the case in practice, and that we encode information sequences of length K bits using a linear block encoder or a convolutional encoder that is zero-terminated such that there are 2^K codewords, of length N bits.

If the code is linear and the channel is symmetric, the probability of error does not depend on the transmitted codeword. Therefore, we can assume that the all-zero codeword $\boldsymbol{c} = \mathbf{0}$Â was transmitted, resulting in

$$\boxed{P_{\mathsf{w}} = \Pr(\hat{\boldsymbol{c}} \neq \mathbf{0}|\mathbf{0}).}$$

If the all-zero codeword $\mathbf{0}$ is transmitted, a decoding error will occur if the decoder selects any of the $2^K - 1$ nonzero codewords, i.e.,

$$\begin{aligned} P_{\mathsf{w}} &= \Pr(\hat{\boldsymbol{c}} \neq \mathbf{0}|\mathbf{0}) \\ &= \Pr\left(\bigcup_{\boldsymbol{c} \neq \mathbf{0}} \hat{\boldsymbol{c}} = \boldsymbol{c}|\mathbf{0} \right). \end{aligned}$$

We can now use the union bound (6.8) and upper bound the probability of the union of events by the sum of their probabilities,

$$\begin{aligned} P_{\mathsf{w}} &= \Pr\left(\bigcup_{\boldsymbol{c}\neq\boldsymbol{0}} \hat{\boldsymbol{c}} = \boldsymbol{c}|\boldsymbol{0}\right) \\ &\leq \sum_{\boldsymbol{c}\neq\boldsymbol{0}} \Pr(\hat{\boldsymbol{c}} = \boldsymbol{c}|\boldsymbol{0}) \\ &= \sum_{\boldsymbol{c}\neq\boldsymbol{0}} \Pr(\boldsymbol{0}\to\boldsymbol{c}), \end{aligned} \tag{9.19}$$

where we have defined $\Pr(\boldsymbol{0}\to\boldsymbol{c}) \triangleq \Pr(\hat{\boldsymbol{c}} = \boldsymbol{c}|\boldsymbol{0})$ as the probability that the decoder selects the codeword $\hat{\boldsymbol{c}} = \boldsymbol{c}$ if the all-zero codeword $\boldsymbol{0}$ was transmitted. The probability $\Pr(\boldsymbol{0}\to\boldsymbol{c})$ is usually referred to as the **pairwise error probability**.

Note that $\Pr(\boldsymbol{0}\to\boldsymbol{c})$ depends only on the Hamming distance between $\boldsymbol{0}$ and $\boldsymbol{c}$, or equivalenty on the Hamming weight of $\boldsymbol{c}$ (since $d_{\mathsf{H}}(\boldsymbol{c},\boldsymbol{0}) = w_{\mathsf{H}}(\boldsymbol{c}+\boldsymbol{0}) = w_{\mathsf{H}}(\boldsymbol{c})$, see (7.4)), and not on the particular codeword $\boldsymbol{c}$. Therefore, denoting by $\boldsymbol{c}_d$ a codeword of Hamming weight d, we can rewrite (9.19) as

$$\begin{aligned} P_{\mathsf{w}} &\leq \sum_{\boldsymbol{c}\neq\boldsymbol{0}} \Pr(\boldsymbol{0}\to\boldsymbol{c}) \\ &= \sum_{d=d_{\mathsf{min}}}^{N} A_d \Pr(\boldsymbol{0}\to\boldsymbol{c}_d), \end{aligned} \tag{9.20}$$

where A_d is the number of codewords of Hamming weight d.[4] $\{A_d\}$, $d = 1,\ldots,N$, is usually referred to as the **weight enumerator** or the **distance spectrum** of the code.

In the following, we derive the pairwise error probability $\Pr(\boldsymbol{0}\to\boldsymbol{c}_d)$ for hard-decision decoding and soft-decision decoding.

9.7.1 Hard-Decision Decoding

We derive $\Pr(\boldsymbol{0}\to\boldsymbol{c}_d)$ in (9.20), i.e., the probability that the decoder decodes onto a codeword of weight d, $\boldsymbol{c}_d$, if the all-zero codeword $\boldsymbol{0}$ was transmitted. We have to distinguish between two cases, d being odd and d being even,

- d odd: In this case the decoder will make an error if the channel introduces $\lceil \frac{d}{2} \rceil$ or more errors such that the received word $\bar{\boldsymbol{y}}$ is closer to codeword $\boldsymbol{c}_d$ than to $\boldsymbol{0}$. In other words, if the channel introduces $\lceil \frac{d}{2} \rceil$ or more errors in the d positions corresponding to the ones of codeword $\boldsymbol{c}_d$. Assume that the number of errors introduced by the channel is j (note that j must satisfy $\lceil \frac{d}{2} \rceil \leq j \leq d$). We can place j errors in d positions in $\binom{d}{j}$ possible ways, and this error will have a probability $\varepsilon^j(1-\varepsilon)^{d-j}$, where ε is the crossover probability of the BSC. Therefore, for d being odd,

$$\Pr(\boldsymbol{0}\to\boldsymbol{c}_d) = \sum_{j=\lceil \frac{d}{2} \rceil}^{d} \binom{d}{j} \varepsilon^j (1-\varepsilon)^{d-j}. \tag{9.21}$$

- d even: In this case the decoder will make an error if the channel introduces $\frac{d}{2}+1$ or more errors in the d positions corresponding to the ones of $\boldsymbol{c}_d$ such that the received word $\bar{\boldsymbol{y}}$Â is closer to $\boldsymbol{c}_d$ than to $\boldsymbol{0}$.

[4] A_d can be obtained from the trellis diagram if the trellis is small (i.e., for short L). For large trellises, a computer search is required.

If the channel introduces exactly $\frac{d}{2}$ errors in these positions, then the decoder will make an error with probability 1/2. Therefore, for d even,

$$\Pr(\mathbf{0} \to \boldsymbol{c}_d) = \frac{1}{2}\binom{d}{d/2}\varepsilon^{d/2}(1-\varepsilon)^{d/2} + \sum_{j=\frac{d}{2}+1}^{d}\binom{d}{j}\varepsilon^{j}(1-\varepsilon)^{d-j}. \tag{9.22}$$

Finally, using (9.21) and (9.22) in (9.20), we obtain the bound on the word error probability for hard-decision decoding.

9.7.2 Soft-Decision Decoding

We derive $\Pr(\boldsymbol{c}_d \to \mathbf{0})$ for soft-decision decoding for transmission over the AWGN channel, where the Gaussian noise has zero mean and variance $\sigma^2 = \mathsf{N}_0/2$. Observe that if the all-zero codeword $\mathbf{0}$ is transmitted, the decoder wrongly selects codeword $\boldsymbol{c}_d$ instead of $\mathbf{0}$ if $d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{y}) < d_{\mathsf{E}}(\boldsymbol{x}(\mathbf{0}), \boldsymbol{y})$ (see Section 9.5.2), where $\boldsymbol{x}(\boldsymbol{c})$ is the BPSK-modulated sequence corresponding to codeword $\boldsymbol{c}$, i.e.,

$$\Pr(\mathbf{0} \to \boldsymbol{c}_d) = \Pr(d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{y}) < d_{\mathsf{E}}(\boldsymbol{x}(\mathbf{0}), \boldsymbol{y})).$$

Let $\mathsf{E_s}$ the average energy per transmitted symbol and consider the mapping $x(c_i) = (-1)^{c_i}\sqrt{\mathsf{E_s}}$. For example, for the all-zero codeword $\boldsymbol{c} = \mathbf{0}$, we get $\boldsymbol{x}(\mathbf{0}) = (+\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, \ldots)$, and for $\boldsymbol{c} = (1,1,1,0,0,\ldots)$, $\boldsymbol{x}(\boldsymbol{c}) = (-\sqrt{\mathsf{E_s}}, -\sqrt{\mathsf{E_s}}, -\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, \ldots)$. It is easy to see that the Euclidean distance between $\boldsymbol{x}(\boldsymbol{c}_d)$ and $\boldsymbol{x}(\mathbf{0})$ is

$$d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0})) = 2\sqrt{d\mathsf{E_s}}.$$

Since the Gaussian noise has variance $\sigma^2 = \mathsf{N}_0/2$ in all directions, $\Pr(\mathbf{0} \to \boldsymbol{c}_d)$ is the probability that the noise in the direction of $\boldsymbol{c}_d$ (recall that $\boldsymbol{c}_d$ corresponds to a point in an N-dimensional space) has magnitude greater than

$$\frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2} = \sqrt{d\mathsf{E_s}}, \tag{9.23}$$

i.e.,

$$\begin{aligned}
\Pr(\mathbf{0} \to \boldsymbol{c}_d) &= \Pr(d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{y}) < d_{\mathsf{E}}(\boldsymbol{x}(\mathbf{0}), \boldsymbol{y})) \\
&= \Pr\left(\tilde{Y} > \frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2}\right)\Bigg|_{\tilde{Y}\sim\mathcal{N}(0,\sigma^2)} \\
&\overset{(a)}{=} \mathsf{Q}\left(\frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2\sigma}\right) \\
&\overset{(b)}{=} \mathsf{Q}\left(\sqrt{\frac{2d\mathsf{E_s}}{\mathsf{N}_0}}\right) \\
&= \mathsf{Q}\left(\sqrt{\frac{2dR_{\mathsf{c}}\mathsf{E_b}}{\mathsf{N}_0}}\right),
\end{aligned}$$

where $\tilde{y}$ is the projection of $\boldsymbol{y}$ onto the straight line between $\boldsymbol{x}(\mathbf{0})$ and $\boldsymbol{x}(\boldsymbol{c}_d)$, in (a) we used the definition of the Q-function (see (6.9)), and in (b) we used (9.23).

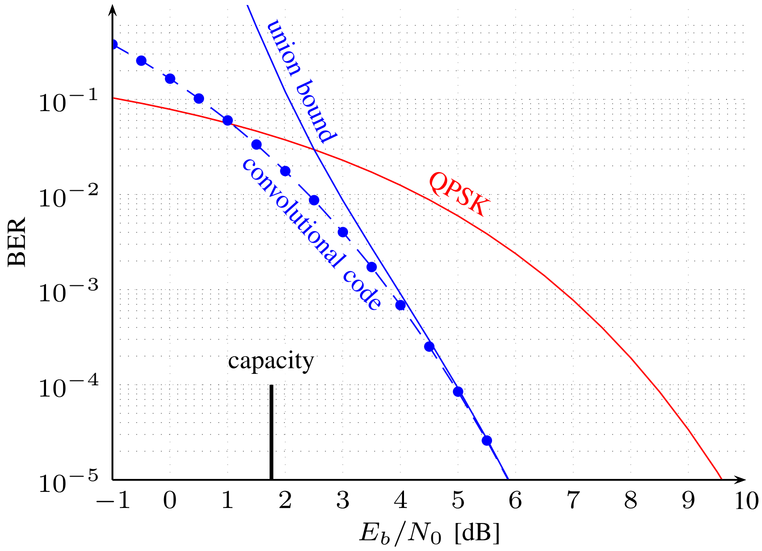


Figure 9.6: Upper bound on the bit error probability (blue line) and simulation results (dotted-line) for the 4-state convolutional code of Figure 9.1, and comparison with uncoded QPSK and the capacity.

The word error probability for soft-decision decoding can then be upper bounded as

$$P_{\mathsf{w}}^{\mathsf{SOFT}} \leq \sum_{d=d_{\mathsf{min}}}^{N} A_d \Pr(\mathbf{0} \to \boldsymbol{c}_d)$$

$$= \sum_{d=d_{\mathsf{min}}}^{N} A_d \mathsf{Q}\left(\sqrt{\frac{2dR_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right) \tag{9.24}$$

For high $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$, the performance is dominated by the term with minimum Hamming distance and we can approximate $P_{\mathsf{w}}^{\mathsf{SOFT}}$ as

$$P_{\mathsf{w}}^{\mathsf{SOFT}} \approx A_{d_{\mathsf{min}}} \mathsf{Q}\left(\sqrt{\frac{2d_{\mathsf{min}}R_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right)$$

The bit error probability P_{b} can be computed in a similar way. We denote by $A_{w,d}$ the number of codewords of weight d produced by an information word of weight w (note that $\sum_{w=1}^{K} A_{w,d} = A_d$). $A_{w,d}$ is usually referred to as the **input-output weight enumerator** of the code. The bit error probability P_{b} can then be upper bounded as

$$P_{\mathsf{b}}^{\mathsf{SOFT}} \leq \frac{1}{K} \sum_{d=d_{\mathsf{min}}}^{N} \sum_{w=1}^{K} w A_{w,d} \mathsf{Q}\left(\sqrt{\frac{2dR_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right). \tag{9.25}$$

In Figure 9.6 we plot the bound in (9.25) on the bit error probability for the 4-state convolutional code of Figure 9.1 with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1+\mathsf{D}+\mathsf{D}^2 \; 1+\mathsf{D}^2)$. We also plot the actual bit error rate obtained by simulating the code. We observe that the upper bound is very accurate for high $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$, while it diverges for low $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$. As a comparison, we also plot the curve for uncoded BPSK. For low values of $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$, uncoded transmission performs better, since as we already discussed in Section 7.5, for such low values of $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$ the code is not strong enough to compensate for the loss in energy per transmitted BPSK symbol due to the use of coding (note that for uncoded transmission $\mathsf{E}_{\mathsf{s}} = \mathsf{E}_{\mathsf{b}}$, while for coded transmission $\mathsf{E}_{\mathsf{s}} = \mathsf{E}_{\mathsf{b}}R_{\mathsf{c}} < \mathsf{E}_{\mathsf{b}}$).

However, coding is advantageous for moderate-to-high $\mathsf{E_b}/\mathsf{N_0}$. To achieve a probability of error of 10^{-5} it requires around 5.9 dB compared to 9.5 dB for uncoded transmission, i.e., this simple code brings a gain of 3.6 dB compared to uncoded transmission, at the expense of lowering the transmission rate.

Remark 9.2 From (9.24) and (9.25) we observe that the word error probability depends only on the set of codewords, i.e., on the code, but not on the way information words are mapped to codewords. On the other hand, the bit error probability depends on how information words are mapped to codewords, i.e., it depends on the encoder.

9.8 Coding Gain

We have already discussed in Section 7.5 that the advantage of coding is measured in terms of **coding gain**, defined as the difference (in dB) in the required $\mathsf{E_b}/\mathsf{N_0}$ to achieve a given bit error probability P_b between uncoded and coded transmission,

$$\boxed{\mathsf{G_c} = 10\log_{10}\left(\frac{\frac{\mathsf{E_b}}{\mathsf{N_0}}\Big|_\mathsf{unc}}{\frac{\mathsf{E_b}}{\mathsf{N_0}}\Big|_\mathsf{cod}}\right)}$$

For very high $\mathsf{E_b}/\mathsf{N_0}$, i.e., $\mathsf{E_b}/\mathsf{N_0} \to \infty$, the asymptotic coding gain is

$$\boxed{\mathsf{G}_{\mathsf{c},\infty} = 10\log_{10}(R_\mathsf{c} d_\mathsf{min}),}$$

which can be easily obtained by approximating the bit error probability in (9.25) by considering only the term at minimum Hamming distance and setting the multiplicity to one.

Example 9.6 It is easy to see that the convolutional code resulting from the encoders in Figures 9.1 and 9.3 has $d_\mathsf{min} = 5$. Therefore, the asymptotic coding gain over uncoded transmission is $\mathsf{G}_{\mathsf{c},\infty} = 10\log_{10}\left(\frac{1}{2}\cdot 5\right) = 3.979$ dB, i.e., slightly larger than the gain that we observed in Figure 9.6 for $P_\mathsf{b} = 10^{-5}$.

Chapter 10

Turbo Codes

SINCE THE seminal paper by Claude E. Shannon in 1948, coding theorists have tried to devise **practical** coding schemes that approach the channel capacity. By practical we mean codes that can be decoded with reasonable decoding complexity. This turned out to be a formidable task. In the previous chapters, we have seen that block codes and convolutional codes yield significant coding gains with respect to uncoded transmission. However, they still perform very far away from capacity.

A lesson taught by Shannon (see Section 4.6) was that, to approach capacity, very large block lengths are required. The performance of block codes improves indeed with the block length and, correspondingly, the performance of convolutional codes improves with the memory of the encoder. Therefore, to approach capacity one might use, e.g., a convolutional code with a huge number of states. Unfortunately, the decoding complexity of block codes and convolutional codes (with optimum decoding) increases exponentially with the block length and the memory of the encoder, respectively. Thus, classical block codes and convolutional codes cannot approach capacity with reasonable decoding complexity.

For several decades coding theorists believed that it was not possible to approach capacity with low decoding complexity. A breakthrough came in 1993 (45 years after Shannon's channel coding theorem!), when two French Professors, Claude Berrou and Alain Glavieux, introduced a new class of codes, called **turbo codes**, with unprecedented performance: Turbo codes were the first class of codes that approached capacity with low decoding complexity [8]. The introduction of turbo codes represents one of the most important breakthroughs in coding theory and, together with the rediscovery of low-density parity-check (LDPC) codes (discussed in the next chapter), revolutionized the theory and practice of coding theory and gave birth to what now is known as **modern coding theory**. Turbo codes are today an integral part of many communication standards, including 3G/4G mobile communications (e.g., LTE), satellite communication standards such as DVB-RCS, and WiMAX.

In this chapter, we introduce the main concepts of turbo codes and the associated low-complexity iterative decoding algorithm.

10.1 Turbo Codes: Parallel Concatenated Convolutional Codes

The block diagram of a turbo encoder is depicted in Figure 10.1. A turbo encoder is built from the parallel concatenation of two recursive systematic convolutional (RSC) encoders through a pseudo-random interleaver. As such, turbo codes are also referred to as **parallel concatenated convolutional codes**.

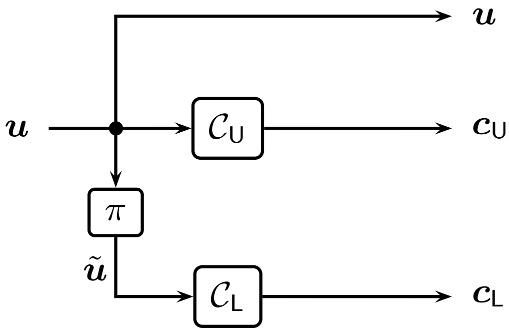

$\boldsymbol{u}$
$\boldsymbol{u}$
$\mathcal{C}_{\mathsf{U}}$
$\boldsymbol{c}_{\mathsf{U}}$
π
$\tilde{\boldsymbol{u}}$
$\mathcal{C}_{\mathsf{L}}$
$\boldsymbol{c}_{\mathsf{L}}$

Figure 10.1: Block diagram of a turbo code. The code is systematic and the rate is $R = 1/3$.

Encoding proceeds as follows. The information vector $\boldsymbol{u}$, of length K bits, is first encoded by a rate-1 RSC encoder, called the **upper encoder**, with generator matrix

$$\boldsymbol{G}_{\mathsf{U}}(\mathsf{D}) = \left(\frac{\boldsymbol{g}_1(\mathsf{D})}{\boldsymbol{g}_2(\mathsf{D})}\right).$$

The vector $\boldsymbol{u}$ is then permuted by a permutor, π, called **interleaver** in the coding jargon, into vector $\tilde{\boldsymbol{u}}$. The permuted vector $\tilde{\boldsymbol{u}}$ is then encoded by another rate-1 RSC encoder, called the **lower encoder**, with generator matrix

$$\boldsymbol{G}_{\mathsf{L}}(\mathsf{D}) = \left(\frac{\boldsymbol{g}_1(\mathsf{D})}{\boldsymbol{g}_2(\mathsf{D})}\right).$$

The upper and lower encoders are also referred to as **component encoders** of the turbo code. The original turbo codes proposed by Berrou and Glavieux considered identical component encoders. However, one may consider different component encoders.

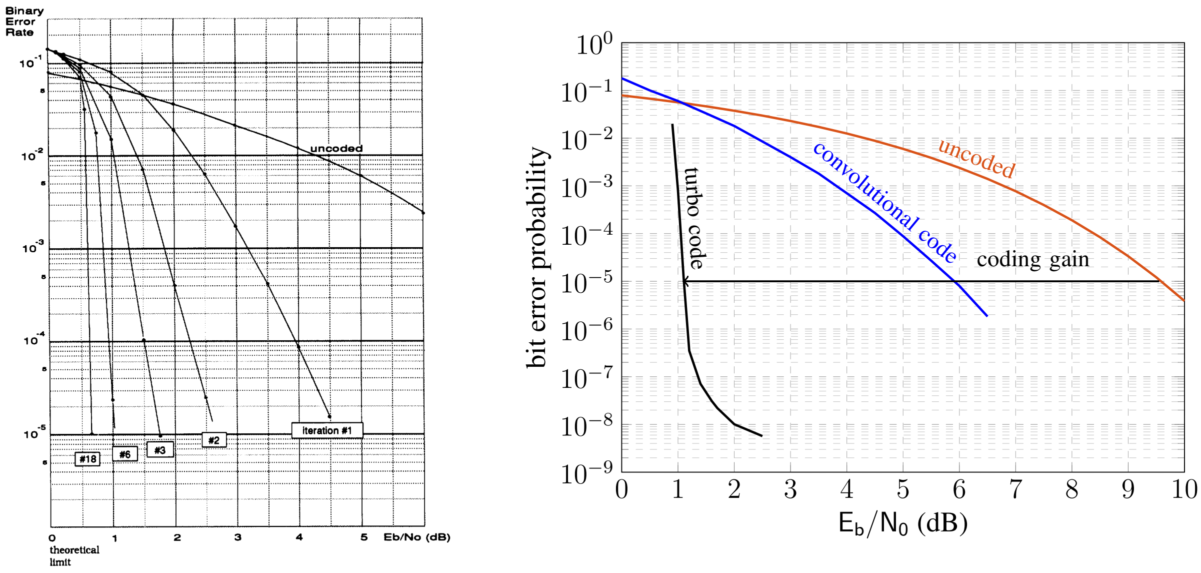
We denote by $\boldsymbol{c}_{\mathsf{U}}$ the codeword of the upper code, $\mathcal{C}_{\mathsf{U}}$, of length K bits. Likewise, we denote by $\boldsymbol{c}_{\mathsf{L}}$ the codeword of the lower code, $\mathcal{C}_{\mathsf{L}}$, of length K bits. The codeword of the turbo code corresponding to the information sequence $\boldsymbol{u}$ is obtained as $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{c}_{\mathsf{U}}, \boldsymbol{c}_{\mathsf{L}})$, i.e., as the concatenation of the information word and the codewords of the upper and lower component encoders. Therefore, the resulting turbo code is a systematic code (see Definition 8.1) and (neglecting termination of the component encoders) the code rate is

$$R_{\mathsf{c}} = \frac{K}{3K} = \frac{1}{3}.$$

Higher rates can be obtained by periodically deleting some bits at the output of the turbo encoder, which are therefore not transmitted, according to a given pattern. This way of obtaining higher rates from lower rate codes is referred to as **code puncturing**. On the other hand, lower code rates can be achieved by using lower rate encoders as component encoders or by concatenating more encoders in parallel.

Example 10.1 To achieve a rate-1/2 turbo code, one may puncture every second bit at the output of the upper and lower encoders. If $\boldsymbol{u}$ is of length K bits, the codeword of the (unpunctured) turbo code, $(\boldsymbol{u}, \boldsymbol{c}_{\mathsf{U}}, \boldsymbol{c}_{\mathsf{L}})$, is of length $K + K + K = 3K$ bits. Now, if we puncture every other bit of $\boldsymbol{c}_{\mathsf{U}}$ and $\boldsymbol{c}_{\mathsf{U}}$ we obtain a codeword $(\boldsymbol{u}, \boldsymbol{c}_{\mathsf{U}}^{\mathsf{p}}, \boldsymbol{c}_{\mathsf{L}}^{\mathsf{p}})$ of length $K + K/2 + K/2 = 2K$, i.e., the resulting turbo code has rate $R_{\mathsf{c}} = K/2K = 1/2$.

The performance curve of the original turbo code is depicted in Figure 10.2(a) (the figure has been borrowed from the original paper by Berrou, Glavieux, and Thitimajshima [8]), where we plot the bit error



(a)

(b)

Figure 10.2: (a) BER performance of the original turbo code. $R_\mathsf{c} = 1/2$ and $K = 65536$. (The figure has been taken from the original paper [8]) (b) Typical BER curve of a turbo code, characterized by a waterfall region and an error floor region. For this example, $R_\mathsf{c} = 1/2$ and $K = 12288$, and the component encoders are 8-state encoders with generator matrix $\boldsymbol{G}(\mathsf{D}) = \left(\frac{1+\mathsf{D}+\mathsf{D}^3}{1+\mathsf{D}^2+\mathsf{D}^3}\right)$. This is the turbo code of the 3GPP-LTE standard. The convolutional code is the $R_\mathsf{c} = 1/2$ convolutional code of Figure 9.1.

rate (BER) as a function of $\mathsf{E_b}/\mathsf{N_0}$. The component encoders are 16-state convolutional encoders with generator matrix $\boldsymbol{G}(\mathsf{D}) = \left(\frac{1+\mathsf{D}^4}{1+\mathsf{D}+\mathsf{D}^2+\mathsf{D}^3+\mathsf{D}^4}\right)$. The information block length is $K = 65536$ bits and the code rate is $R_\mathsf{c} = 1/2$, which is obtained by puncturing every other bit of the codewords $\boldsymbol{c}_\mathsf{U}$ and $\boldsymbol{c}_\mathsf{L}$. Note that for 18 decoding iterations the code performs only 0.5 dB away from capacity (0.187 dB) at a BER of 10^{-5}! (as a comparison, the $R_\mathsf{c} = 1/2$ 4-state convolutional code in Figure 9.6 in Section 9.7.2 performs 4 dB away from capacity). At the time of the publication of Berrou *et al.* seminal paper, this performance was so astonishing, much better than any practical code proposed up to that date, that the scientific community was very skeptical about the results.

From Figure 10.2(a) we may interpret that the turbo code can achieve extremely low error probabilities. In reality, Figure 10.2(a) does not show the full picture. The characteristic BER curve of a turbo code is shown in Figure 10.2(b). The performance curve of a turbo code is characterized by two well-defined regions. In the first one, called **waterfall** region, the BER decreases sharply with $\mathsf{E_b}/\mathsf{N_0}$. The curve then flattens out in the so-called **error floor** region. The error floor is determined by the minimum Hamming distance of the code. The fact that the error floor is not observed in Figure 10.2(a) is because the error floor appears below 10^{-6}. Typically, to achieve relatively low error floors ($10^{-5} - 10^{-8}$), 8-state or 16-state component encoders and information word lengths of a few thousand bits suffice.

10.2 The Need of Recursive Encoders

When considering convolutional codes as standalone codes, there is no reason to consider recursive encoders. Indeed, for a given code rate and memory, feedforward convolutional encoders are better than recursive convolutional encoders, in the sense that their input-output weight enumerator (see Section 9.7.2) is slightly better, hence they yield slightly lower BER.

However, recursive encoders play a crucial role in turbo codes. To show why, consider the following. Assume that the information word $\boldsymbol{u}$ is of weight one. This will generate a codeword $\boldsymbol{c}_{\mathsf{U}}$ of $\mathcal{C}_{\mathsf{U}}$ of Hamming weight at most $w_{\mathsf{H}}(\boldsymbol{c}_{\mathsf{U}}) \leq \nu+1$, where ν is the memory of the encoder (see Chapter 9). Clearly, the Hamming weight of the permuted codeword $\tilde{\boldsymbol{u}}$ is also one. Therefore, $\tilde{\boldsymbol{u}}$ will generate a codeword $\boldsymbol{c}_{\mathsf{L}}$ of $\mathcal{C}_{\mathsf{L}}$ of Hamming weight $w_{\mathsf{H}}(\boldsymbol{c}_{\mathsf{L}}) \leq \nu+1$. Overall, the minimum Hamming distance of the turbo code is upperbounded by

$$d_{\mathsf{min}} \leq 1 + 2(\nu+1),$$

where the first term of the summation accounts for the Hamming weight of the information word (the minimum Hamming weight of information words (except the all-zero word) is one, since there are information words with a single one). This is a rather poor minimum Hamming distance. Furthermore, the bound above is independent of the interleaver size. In other words, there is nothing to gain by increasing the code length!

Example 10.2 Consider the rate-1/3 turbo code built from the parallel concatenation of two 4-state feedforward component encoders with identical generator matrix

$$\boldsymbol{G}(\mathsf{D}) = \left(1 + \mathsf{D} + \mathsf{D}^2\right).$$

The codeword at the output of each component encoder generated by a weight-1 information word $\boldsymbol{u}(\mathsf{D}) = \mathsf{D}^j$ is

$$\boldsymbol{x}(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{G}(\mathsf{D}) = \mathsf{D}^j(1 + \mathsf{D} + \mathsf{D}^2) = \mathsf{D}^j + \mathsf{D}^{j+1} + \mathsf{D}^{j+2},$$

i.e., a weight-1 information word generates a codeword of weight 3 at the output of each component encoder. Overall, a weight-1 information word generates a codeword of the turbo code of weight $1+3+3 = 7$, independently of the interleaver size. The minimum Hamming distance of the turbo code that we have built is indeed 7. But now consider that the best rate-1/3, 4-state convolutional code has minimum distance $d_{\mathsf{min}} = 8$. Therefore, the turbo code in this example extremely poor minimum distance. The reason for that is the use of feedforward encoders as component encoders.

Let us consider now the use of RSC component encoders. In this case, weight-1 information words are not a problem anymore. In fact, a weight-1 information word will generate an infinite weight codeword at the output of each component encoder, since D^j is not divisible by a nontrivial polynomial $\boldsymbol{g}_1(\mathsf{D})$.[1]

Example 10.3 Consider the rate-1/3 turbo code built from the parallel concatenation of two 4-state RSC component encoders with identical generator matrix

$$\boldsymbol{G}(\mathsf{D}) = \left[\frac{1 + \mathsf{D}^2}{1 + \mathsf{D} + \mathsf{D}^2}\right].$$

The codeword at the output of each component encoder generated by a weight-1 information word $\boldsymbol{u}(\mathsf{D}) = \mathsf{D}^j$ is

$$\boldsymbol{x}(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{G}(\mathsf{D}) = D^j \frac{1 + \mathsf{D}^2}{1 + \mathsf{D} + \mathsf{D}^2} = \mathsf{D}^j \cdot \big(1 + \mathsf{D} + \mathsf{D}^2 + \mathsf{D}^4 + \mathsf{D}^5 + \mathsf{D}^7 + \mathsf{D}^8 + \mathsf{D}^{10} + \mathsf{D}^{11} + \ldots,$$

[1] We remark that a weight-1 information word will generate a codeword of infinite weight only for unterminated component encoders. In practice, since the code length is finite (i.e., we terminate the encoders), a weight-1 information word will generate a codeword of large weight with high probability.

i.e., a weight-1 information word generates a codeword of infinite weight.

10.3 The Role of the Interleaver

We have seen in previous chapters that the performance of a code is dominated by its minimum Hamming distance and, more in general, by its weight enumerator. The main role of the interleaver is therefore to ensure that the turbo code has a large minimum Hamming distance. The main idea is the following: If the information sequence $\boldsymbol{u}$ is such that it produces a codeword $\boldsymbol{c}_{\mathsf{U}}$Â of low weight at the output of the upper encoder, the role of the interleaver is to permute $\boldsymbol{u}$ in such a way that the codeword $\boldsymbol{c}_{\mathsf{L}}$Â at the output of the lower encoder is of large weight.

The interleaver should pay particular attention to weight-2 information words (weight-1 words are not a problem if recursive encoders are used, as we have seen in the previous section), which tend to yield low-weight codewords, in particular if the two ones are close to each other. A good design rule is therefore to guarantee that if two input bits of $\boldsymbol{u}$ in positions i and j are within s positions to each other, i.e., $|i-j| \leq s$, then in the permuted word $\tilde{\boldsymbol{u}}$ they should be spread further apart, i.e., $|\pi(i)-\pi(j)| > s$. For a given information block length K, s should be chosen as large as possible. This ensures that if for a weight-2 information word $\boldsymbol{u}$ the two ones are close to each other, and thus likely generate a low-weight codeword $\boldsymbol{c}_{\mathsf{U}}$, then the two ones in $\tilde{\boldsymbol{u}}$ will be far apart from each other, and thus will likely generate a high-weight codeword $\boldsymbol{c}_{\mathsf{L}}$.

10.4 Decoding Turbo Codes: Iterative (Turbo) Decoding

Ideally, one would like to decode turbo codes optimally. We start from the optimum decoding rule, i.e., the MAP rule,

$$\hat{u}_i = \arg\max_{u_i} p(u_i|\boldsymbol{y}). \tag{10.1}$$

Note that the MAP decoding rule above is slightly different from the MAP rule discussed in previous chapters (see for example (6.1)). The reason is that turbo codes focus on the minimization of the bit error probability, rather than the word error probability. Therefore, rather than choosing among all codewords (equivalently, all information words) the one that maximizes the word-wise a posteriori probability (APP) $p(\boldsymbol{u}|\boldsymbol{y})$, to minimize the bit error probability, for each information bit u_i we need to select among all information bits the one that maximizes the bit-wise APP $p(u_i|\boldsymbol{y})$.

The goal of the turbo decoder is therefore to compute the APPs $p(u_i|\boldsymbol{y})$ based on the received (noisy) sequence $\boldsymbol{y} = (\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}})$, where $\boldsymbol{y}^{\boldsymbol{u}}$, $\boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}$, and $\boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}}$ are the received sequences corresponding to the information sequence $\boldsymbol{u}$ and the codewords $\boldsymbol{c}_{\mathsf{U}}$ and $\boldsymbol{c}_{\mathsf{L}}$ of the upper and lower encoders, respectively. For convenience, we will write the APP $p(u_i|\boldsymbol{y})$ as $P_{\mathsf{APP}}(u_i|\boldsymbol{y})$, making it explicit that it is an APP. The decoding rule is then

$$\hat{u}_i = \begin{cases} 1 & \text{if } P_{\mathsf{APP}}(u_i=1|\boldsymbol{y}) > P_{\mathsf{APP}}(u_i=0|\boldsymbol{y}) \\ 0 & \text{if } P_{\mathsf{APP}}(u_i=0|\boldsymbol{y}) < P_{\mathsf{APP}}(u_i=1|\boldsymbol{y}) \end{cases}. \tag{10.2}$$

For implementation reasons, it is usually more convenient to work with so-called log-likelihood ratios (LLRs),

$$L_{\mathsf{APP}}(u_i|\boldsymbol{y}) \triangleq \ln \frac{P_{\mathsf{APP}}(u_i=0|\boldsymbol{y})}{P_{\mathsf{APP}}(u_i=1|\boldsymbol{y})}.$$

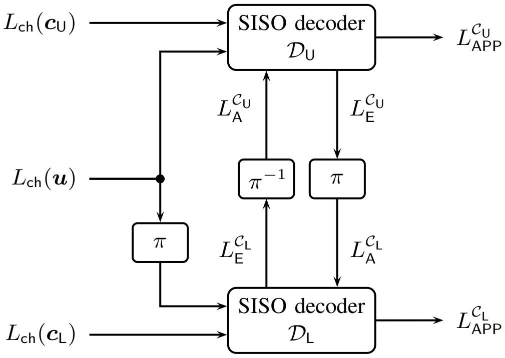

$L_{\mathsf{ch}}(\boldsymbol{c}_{\mathsf{U}})$
SISO decoder
$\mathcal{D}_{\mathsf{U}}$
$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}$
$L_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}$
$L_{\mathsf{E}}^{\mathcal{C}_{\mathsf{U}}}$
$L_{\mathsf{ch}}(\boldsymbol{u})$
π^{-1}
π
π
$L_{\mathsf{E}}^{\mathcal{C}_{\mathsf{L}}}$
$L_{\mathsf{A}}^{\mathcal{C}_{\mathsf{L}}}$
SISO decoder
$\mathcal{D}_{\mathsf{L}}$
$L_{\mathsf{ch}}(\boldsymbol{c}_{\mathsf{L}})$
$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{L}}}$

Figure 10.3: Iterative (turbo) decoder for the rate-1/3 turbo code of Figure 10.1.

Using LLRs the decoding rule in (10.2) can be rewritten as

$$\hat{u}_i = \begin{cases} 1 & \text{if } L_{\mathsf{APP}}(u_i|\boldsymbol{y}) < 0 \\ 0 & \text{if } L_{\mathsf{APP}}(u_i|\boldsymbol{y}) > 0 \end{cases}.$$

Unfortunately, optimum MAP decoding of turbo codes, i.e., computing the log-APPs $L_{\mathsf{APP}}(u_i|\boldsymbol{y})$ exactly, is unfeasible.[2] To decode turbo codes with reasonable complexity, Berrou and Glavieux proposed a low-complexity, suboptimal iterative decoding algorithm, known as **turbo decoding**, to estimate the log-APPs $L_{\mathsf{APP}}(u_i|\boldsymbol{y})$. Despite being suboptimal, it achieves remarkable performance. The idea is to break the decoding of the turbo code into the cascade of two simpler decoders, i.e., the decoders of the two component encoders. Then, the component decoders exchange information about the reliability of their estimates, usually referred to as **soft information**, in an iterative manner. In simple words, the two decoders **cooperate** in estimating the transmitted information word $\boldsymbol{u}$. As already observed in Figure 10.2(a), the reliability of the estimation increases with the number of iterations.

The decoder of a turbo code is depicted in Figure 10.3. It consists of two so-called **soft-input soft-output** (SISO) decoders, one for each of the component encoders, which exchange soft information iteratively. Each of the SISO decoders performs optimum MAP decoding (see (10.1)) of the corresponding component encoder. In particular, the decoder $\mathcal{D}_{\mathsf{U}}$ (corresponding to encoder $\mathcal{C}_{\mathsf{U}}$) computes the log-APPs

$$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}) = \ln \frac{P_{\mathsf{APP}}(u_i = 0|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})}{P_{\mathsf{APP}}(u_i = 1|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})}$$

for each information bit u_i and the decoder $\mathcal{D}_{\mathsf{L}}$ (corresponding to encoder $\mathcal{C}_{\mathsf{L}}$) computes the log-APPs

$$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{L}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}}) = \ln \frac{P_{\mathsf{APP}}(u_i = 0|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}})}{P_{\mathsf{APP}}(u_i = 1|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}})}.$$

Note that the two decoders work with different channel observations. The final decision can then be taken based on the log-APPs $L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})$ and $L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{L}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{L}}})$.

The SISO decoder and the iterative decoding algorithm are described in detail in the following subsection.

Remark 10.1 Due to lack of time, in this course we will not address the computation of the log-APPs $L_{\mathsf{APP}}(u_i|\boldsymbol{y})$. The APPs can be computed in an efficient way using the BCJR algorithm, named after its inventors Bahl, Cocke, Jelinek and Raviv. Similar to the Viterbi decoding algorithm, the BCJR algorithm works on the trellis of the code. However, contrary to the Viterbi algorithm, which makes hard decisions,

[2] Turbo codes can be represented by a trellis diagram, whose state complexity, although difficult to evaluate exactly, increases exponentially with the interleaver size.

it generates soft information at the output. For details on the BCJR algorithm, the interested reader is referred to, e.g., [9, Sec. 4.5.3].

10.4.1 The Soft-Input Soft-Output Decoder

Let us consider the SISO decoder $\mathcal{D}_{\mathsf{U}}$ for the upper component encoder $\mathcal{C}_{\mathsf{U}}$ in Figure 10.3. The SISO decoder has three inputs. The first two inputs correspond to the channel observations, in the form of LLRs, of the information bits $\boldsymbol{u}$ and the code bits $\boldsymbol{c}_{\mathsf{U}}$. The channel LLR for the i-th information bit is given by

$$L_{\mathsf{ch}}(y_i^u|u_i) = \ln \frac{P(y_i^u|u_i=0)}{P(y_i^u|u_i=1)}.$$

Likewise, the channel LLR for the i-th code bit of codeword $\boldsymbol{c}_{\mathsf{U}}$ is

$$L_{\mathsf{ch}}(y_i^{x_{\mathsf{u}}}|x_{\mathsf{u},i}) = \ln \frac{P(y_i^{x_{\mathsf{u}}}|x_{\mathsf{u},i}=0)}{P(y_i^{x_{\mathsf{u}}}|x_{\mathsf{u},i}=1)}.$$

The third input corresponds to **a priori** information on the information bits,

$$L_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i) = \ln \frac{P_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i=0)}{P_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i=1)},$$

which is provided by the companion decoder $\mathcal{D}_{\mathsf{L}}$. This will become clearer below.

The SISO decoder $\mathcal{D}_{\mathsf{U}}$ has two outputs. The first output comprises the log-APPs of the information bits,

$$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}) = \ln \frac{P_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i=0|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})}{P_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i=1|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})}. \tag{10.3}$$

Applying Bayes' rule, (10.3) can be rewritten as

$$L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}) = \ln \frac{P(\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}|u_i=0)}{P(\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}|u_i=1)} + \ln \frac{P(u_i=0)}{P(u_i=1)} \tag{10.4}$$

$$= \ln \frac{P^{\mathcal{C}_{\mathsf{U}}}(\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}|u_i=0)}{P^{\mathcal{C}_{\mathsf{U}}}(\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}|u_i=1)} + \ln \frac{P_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i=0)}{P_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i=1)}, \tag{10.5}$$

where the second term in (10.4) is the a priori information on the transmitted bit that the upper decoder $\mathcal{D}_{\mathsf{U}}$Â has and in (10.5) we included the superindex $\mathcal{C}_{\mathsf{U}}$ to stress the fact that these probabilities are for the upper decoder. For conventional decoders, e.g., that of a standalone convolutional code, the second term in (10.4) (equivalently (10.5)) is zero, since typically bits are equiprobable, i.e., $P(u_i = 0) = P(u_i = 1) = 0.5$. However, for **iterative** decoders, each component decoder computes its own APPs $P_{\mathsf{APP}}(u_i = 0)$ and $P_{\mathsf{APP}}(u_i = 1)$. In the case of turbo codes, both $\mathcal{D}_{\mathsf{U}}$ and $\mathcal{D}_{\mathsf{L}}$ compute APPs on the transmitted bits. This information can then be used as a priori information for the decoding of the companion decoder, thus the second term in (10.5) is nonzero (decoder $\mathcal{C}_{\mathsf{L}}$ provides relevant information that we can use!).

In the sequel, to simplify notation, we drop $\boldsymbol{y}^{\boldsymbol{u}}$, $\boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}}$, and y_i^u from $L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i|\boldsymbol{y}^{\boldsymbol{u}}, \boldsymbol{y}^{\boldsymbol{c}_{\mathsf{U}}})$ and $L_{\mathsf{ch}}(y_i^u|u_i)$ and simply write $L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i)$ and $L_{\mathsf{ch}}(u_i)$.

The second output of the decoder is the so-called **extrinsic information**,

$$L_{\mathsf{E}}^{\mathcal{C}_{\mathsf{U}}}(u_i) = L_{\mathsf{APP}}^{\mathcal{C}_{\mathsf{U}}}(u_i) - L_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i) - L_{\mathsf{ch}}(u_i). \tag{10.6}$$

In other words, the extrinsic information of a bit u_i provided by decoder $\mathcal{D}_{\mathsf{U}}$ is obtained by removing the prior knowledge that $\mathcal{D}_{\mathsf{U}}$ has about the bit, $L_{\mathsf{A}}^{\mathcal{C}_{\mathsf{U}}}(u_i)$, and the channel observation $L_{\mathsf{ch}}(u_i)$ from the log-APP

$L^{\mathcal{C}_\mathsf{U}}_{\mathsf{APP}}(u_i)$. This extrinsic information (after interleaving) will be used by the lower decoder $\mathcal{D}_\mathsf{L}$ as a priori information,

$$L^{\mathcal{C}_\mathsf{L}}_{\mathsf{A}}(\tilde{u}_i) = L^{\mathcal{C}_\mathsf{U}}_{\mathsf{E}}(\pi^{-1}(\tilde{u}_i)). \tag{10.7}$$

The role of the extrinsic information will become clear in the following subsection.

In a similar way, we can define $L^{\mathcal{C}_\mathsf{L}}_{\mathsf{A}}(u_i)$, $L^{\mathcal{C}_\mathsf{L}}_{\mathsf{APP}}(u_i)$, and $L^{\mathcal{C}_\mathsf{L}}_{\mathsf{A}}(u_i)$ for the lower decoder, interchanging $\mathcal{C}_\mathsf{U}$ and $\mathcal{C}_\mathsf{L}$ in the expressions above and using $\boldsymbol{c}_\mathsf{L}$ instead of $\boldsymbol{c}_\mathsf{U}$.

10.4.2 The Turbo Decoding Algorithm

We are now ready to describe the iterative (turbo decoding) algorithm used to decode turbo codes. With reference to Figure 10.3, the algorithm is as follows.

1. Iteration 1.

   (a) Decode $\mathcal{C}_\mathsf{U}$ running decoder $\mathcal{D}_\mathsf{U}$, with inputs $L_\mathsf{ch}(\boldsymbol{u})$ and $L_\mathsf{ch}(\boldsymbol{c}_\mathsf{U})$. $L^{\mathcal{C}_\mathsf{U},(1)}_{\mathsf{A}}(\boldsymbol{u})$ is set to zero (at iteration one there is no a priori information on the information bits), where the superscript (1) indicates iteration 1. The decoder outputs are $L^{\mathcal{C}_\mathsf{U},(1)}_{\mathsf{APP}}(\boldsymbol{u})$ and $L^{\mathcal{C}_\mathsf{U},(1)}_{\mathsf{E}}(\boldsymbol{u})$.

   (b) Decode $\mathcal{C}_\mathsf{L}$ running decoder $\mathcal{D}_\mathsf{L}$, with inputs $L_\mathsf{ch}(\tilde{\boldsymbol{u}})$, $L_\mathsf{ch}(\boldsymbol{c}_\mathsf{L})$, and $L^{\mathcal{C}_\mathsf{L},(1)}_{\mathsf{A}}(\tilde{\boldsymbol{u}}) = L^{\mathcal{C}_\mathsf{U},(1)}_{\mathsf{E}}(\pi^{-1}(\tilde{\boldsymbol{u}}))$, i.e., the extrinsic information provided by $\mathcal{C}_\mathsf{U}$ is used as a priori information (after interleaving).

2. Iteration $\ell > 1$.

   (a) Decode $\mathcal{C}_\mathsf{U}$ running decoder $\mathcal{D}_\mathsf{U}$, with inputs $L_\mathsf{ch}(\boldsymbol{u})$, $L_\mathsf{ch}(\boldsymbol{c}_\mathsf{U})$, and $L^{\mathcal{C}_\mathsf{U},(\ell)}_{\mathsf{A}}(\boldsymbol{u}) = L^{\mathcal{C}_\mathsf{L},(\ell-1)}_{\mathsf{E}}(\pi(\boldsymbol{u}))$, i.e., the extrinsic information provided by $\mathcal{C}_\mathsf{L}$ in the previous iteration is used as a priori information (after de-interleaving). The decoder outputs are $L^{\mathcal{C}_\mathsf{U},(\ell)}_{\mathsf{APP}}(\boldsymbol{u})$ and $L^{\mathcal{C}_\mathsf{U},(\ell)}_{\mathsf{E}}(\boldsymbol{u})$.

   (b) Decode $\mathcal{C}_\mathsf{L}$ running decoder $\mathcal{D}_\mathsf{L}$, with inputs $L_\mathsf{ch}(\tilde{\boldsymbol{u}})$, $L_\mathsf{ch}(\boldsymbol{c}_\mathsf{L})$, and $L^{\mathcal{C}_\mathsf{L},(\ell)}_{\mathsf{A}}(\tilde{\boldsymbol{u}}) = L^{\mathcal{C}_\mathsf{U},(\ell)}_{\mathsf{E}}(\pi^{-1}(\tilde{\boldsymbol{u}}))$.

3. Repeat Step 2 until a maximum number of iterations ℓ_max is reached. Then make decisions on the bits u_i according to

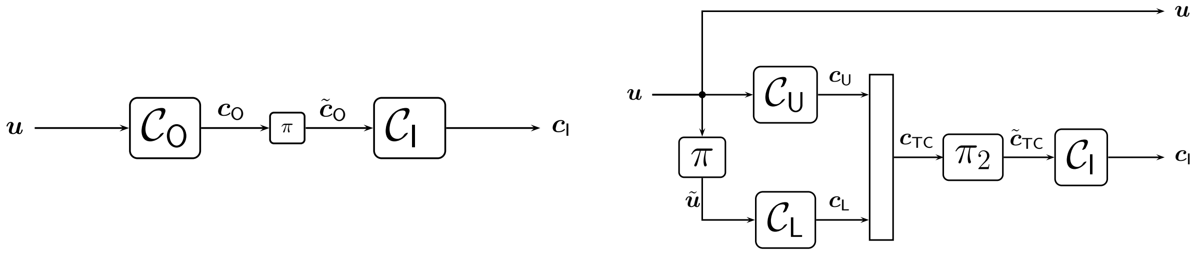
$$\hat{u}_i = \begin{cases} 1 & \text{if } L^{\mathcal{C}_\mathsf{U},(\ell_\mathsf{max})}_{\mathsf{APP}}(u_i) < 0 \\ 0 & \text{if } L^{\mathcal{C}_\mathsf{U},(\ell_\mathsf{max})}_{\mathsf{APP}}(u_i) > 0 \end{cases} .$$

   Alternatively, the decision can be made based on $L^{\mathcal{C}_\mathsf{L},(\ell_\mathsf{max})}_{\mathsf{APP}}$.

Now let's take a look again at Figure 10.2(a), where the BER performance of the original, rate-1/2 turbo code is depicted for $\ell = 1$, 2, 3, 6, and 18 iterations. As can be observed, there is a huge improvement in performance from one to two iterations. Some further gain can be obtained by running more iterations. At some point, however, running more iterations brings no gain. In particular, for this turbo code there is marginal gain beyond 18 iterations. This is due to the fact that the decisions of the two decoders become too correlated at some point so there is not much more to gain by running further iterations. Typically, around $8-10$ iterations are enough to fully exploit the potential of a turbo code.

The use of extrinsic information

One may wonder why the component decoders exchange extrinsic information and not APPs? The reason is that we would like to avoid as much as possible correlation between the decisions of the two decoders. Let's


$\boldsymbol{u}$
$\mathcal{C}_{\mathsf{O}}$
$\boldsymbol{c}_{\mathsf{O}}$
π
$\tilde{\boldsymbol{c}}_{\mathsf{O}}$
$\mathcal{C}_{\mathsf{I}}$
$\boldsymbol{c}_{\mathsf{I}}$
$\boldsymbol{u}$
$\boldsymbol{u}$
$\mathcal{C}_{\mathsf{U}}$
$\boldsymbol{c}_{\mathsf{U}}$
π
$\tilde{\boldsymbol{u}}$
$\mathcal{C}_{\mathsf{L}}$
$\boldsymbol{c}_{\mathsf{L}}$
$\boldsymbol{c}_{\mathsf{TC}}$
π_2
$\tilde{\boldsymbol{c}}_{\mathsf{TC}}$
$\mathcal{C}_{\mathsf{I}}$
$\boldsymbol{c}_{\mathsf{I}}$

(a) Serially concatenated code.

(b) Hybrid concatenated code.

Figure 10.4: Two classes of turbo-like codes.

focus on the lower decoder $\mathcal{D}_{\mathsf{L}}$. Note that the APP generated by decoder $\mathcal{D}_{\mathsf{L}}$ on bit u_i is computed based on the channel observations $L_{\mathsf{ch}}(\boldsymbol{u})$ (for the whole information sequence $\boldsymbol{u}$) and $L_{\mathsf{ch}}(\boldsymbol{c}_{\mathsf{L}})$, as well as on soft information generated by decoder $\mathcal{D}_{\mathsf{U}}$. But now observe that the APPs generated by decoder $\mathcal{D}_{\mathsf{U}}$ also use the knowledge about $L_{\mathsf{ch}}(\boldsymbol{u})$. Since decoder $\mathcal{D}_{\mathsf{L}}$ is already fed with $L_{\mathsf{ch}}(\boldsymbol{u})$ directly, it seems reasonable that the soft information that $\mathcal{D}_{\mathsf{U}}$ passes to $\mathcal{D}_{\mathsf{L}}$ should not contain $L_{\mathsf{ch}}(\boldsymbol{u})$ (otherwise this would correspond to using this information twice at each iteration!), and we should consider passing $L^{\mathcal{C}_{\mathsf{U}}}_{\mathsf{APP}}(u_i) - L_{\mathsf{ch}}(u_i)$ for bit u_i to decoder $\mathcal{D}_{\mathsf{L}}$ instead. However, we are not there yet. We would like to pass truly a priori (independent) information to decoder $\mathcal{D}_{\mathsf{L}}$. But $L^{\mathcal{C}_{\mathsf{U}}}_{\mathsf{APP}}(u_i) - L_{\mathsf{ch}}(u_i)$ includes data from decoder $\mathcal{D}_{\mathsf{L}}$ itself: The a priori information that $\mathcal{D}_{\mathsf{L}}$ passes to $\mathcal{D}_{\mathsf{U}}$! Therefore, what decoder $\mathcal{D}_{\mathsf{U}}$ should pass to decoder $\mathcal{D}_{\mathsf{L}}$ must be

$$L^{\mathcal{C}_{\mathsf{U}}}_{\mathsf{APP}}(u_i) - L^{\mathcal{C}_{\mathsf{U}}}_{\mathsf{A}}(u_i) - L_{\mathsf{ch}}(u_i), \tag{10.8}$$

which is the extrinsic information $L^{\mathcal{C}_{\mathsf{U}}}_{\mathsf{E}}(\boldsymbol{u})$ as defined in (10.6).

10.5 The Rationale Behind Turbo Codes

In Chapters 4 and 5, we have learned that according to Shannon's insights, to achieve capacity very large (infinite, indeed) block lengths are required. Furthermore, Shannon's proof of the channel coding theorem uses a random coding argument. Unfortunately, random codes cannot be decoded in practice, and even for structured codes such as linear block codes, the decoding complexity explodes for large block lengths. Turbo codes address these issues in a very elegant manner:

1. **Randomness.** The idea is to make the code *appear* random while maintaining enough structure to permit decoding. By the presence of the pseudo-random interleaver, turbo codes possess some randomness. However, since the interleaving pattern is known at the receiver, the code can be decoded.

2. **Decoding complexity.** The idea of turbo codes is to build a powerful code that can be decoded in practice by breaking the decoding into simpler steps. The turbo code can be seen as a huge convolutional code, but can be decoded with low complexity (even for lengths of the order of several tens of thousands of bits) using the turbo decoding principle described in the previous section.[3]

[3] The concept of concatenated codes was not introduced in Berrou's paper, but was originally proposed by David Forney in his PhD thesis in the 1970's. However, Forney did not propose the use of iterative decoding.

10.6 Other Code Constructions

The concept of turbo codes, as originally introduced by Berrou, refers to the parallel concatenation of convolutional encoders. However, other concatenated structures are possible. For example, the two component encoders can be concatenated in series, giving rise to serially concatenated convolutional codes [10], shown in Figure 10.4(a). Hybrid constructions, where the component encoders are concatenated both in parallel and in series are also possible [11], see Figure 10.4(b). All these **concatenated** codes can be decoded iteratively in a similar manner as described in Section 10.4 for the classical turbo codes and are usually referred to as **turbo-like codes**.

Chapter 11

Low-Density Parity-Check Codes

Soon after the invention of turbo codes, another class of codes named low-density parity-check (LDPC) codes was rediscovered. LDPC codes were originally introduced by Robert Gallager in 1961 in his PhD thesis. However, despite the fact that LDPC codes had a number of remarkable properties and Gallager introduced in his thesis a suboptimal iterative decoding algorithm to decode them, they were dubbed too complex at that time and went forgotten for several decades (with some exceptions, in particular the important work by Tanner [12]). In 1996, Dave MacKay, a physicist at the University of Cambridge, rediscovered LDPC codes. Nowadays, LDPC codes are one of the most celebrated code constructions and there is still a great deal of research in the coding theory community on this class of codes. As turbo codes, LDPC codes are encountered in most of the current communication standards.

In this chapter, we give a brief introduction to LDPC codes and to its underlying suboptimal iterative decoding algorithm.

11.1 Main Definitions

An LDPC code is a binary linear block code. Therefore, as discussed in Chapter 8, it can be described by a (typically large) parity-check matrix $\boldsymbol{H}$ of dimensions $m \times n$, where n is the code length. As opposed to classical algebraic block codes, an important property of LDPC codes is that its parity-check matrix has a low density of ones, a property which indeed gives name to this class of codes. In practice, if $w_{\mathsf{r},i}$ and $w_{\mathsf{c},i}$ are the weight of the i-th row and the i-th column, respectively, of $\boldsymbol{H}$, a low-density of ones implies that $w_{\mathsf{r},i} \ll n$ and $w_{\mathsf{c},i} \ll m$. The fact that the density of ones in the parity-check matrix is low allows for a low-complexity suboptimal decoding with excellent performance, as we will discuss in Section 11.4.

If the number of ones in each row and each column of $\boldsymbol{H}$ is the same, i.e., if $w_{\mathsf{r},i} = w_{\mathsf{r}}\ \forall i$ and $w_{\mathsf{c},i} = w_{\mathsf{c}}\ \forall i$, the LDPC code is said to be **regular**. In this case, the code rate is given by

$$R_{\mathsf{c}}^{\mathsf{reg}} \geq 1 - \frac{w_{\mathsf{c}}}{w_{\mathsf{r}}}, \tag{11.1}$$

with equality if all rows of $\boldsymbol{H}$ are linearly independent. On the other hand, if the number of ones is not the same for all rows and/or for all columns, the LDPC code is said to be **irregular**. In this case, the code rate can be computed by considering the average row and column weights,

$$R_{\mathsf{c}}^{\mathsf{irreg}} \geq 1 - \frac{\bar{w}_{\mathsf{c}}}{\bar{w}_{\mathsf{r}}}, \tag{11.2}$$

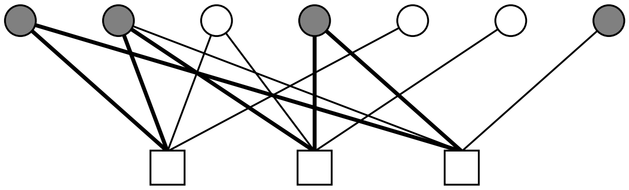


Figure 11.1: Bipartite graph of the $(7,4)$ Hamming code given by the parity-check matrix in (11.4). VNs, corresponding to code bits, are represented by circles, and CNs, corresponding to parity-check equations, are represented by squares. The VNs in gray color are the neighbors of CN c_3. The bold edges show a cycle of length 6.

where

$$\bar{w}_{\mathsf{r}} = \frac{1}{m}\sum_{i=1}^{m} w_{\mathsf{r},i} \qquad \text{and} \qquad \bar{w}_{\mathsf{c}} = \frac{1}{n}\sum_{i=1}^{n} w_{\mathsf{c},i}. \tag{11.3}$$

11.2 Graphical Representation of LDPC Codes: The Bipartite Graph

As an alternative to the representation of the code via the parity-check matrix $\boldsymbol{H}$, LDPC codes can be represented in a graphical, compact form using a **bipartite graph**. The graphical representation of LDPC codes via bipartite graphs was introduced by Tanner in 1981. For this reason, it is also quite common to refer to bipartite graphs as **Tanner graphs**. More generally, any linear block code (not only LDPC codes) can be represented by a bipartite graph.

The Tanner graph of a linear block code is a bipartite graph that consists of two types of nodes, **variable nodes** (VNs), which represent the code bits, and **check nodes** (CNs), also called constraint nodes, which represent the parity-check equations that the code bits satisfy. Therefore, the bipartite graph has n VNs and m CNs. In the graph, there exist an edge e between a VN and a CN if the code bit corresponding to the VN participates in the parity-check equation represented by the CN. It will also be very useful to define the **neighborhood** of a VN node and of a CN node: The neighborhood of a VN v, denoted by $\mathcal{N}(\mathsf{v})$, is the set of all CNs it is connected to. Likewise, the neighborhood of a CN c, denoted by $\mathcal{N}(\mathsf{c})$, is the set of all VNs it is connected to. Let's clarify this by a simple example.

Example 11.1 Consider the $(7,4)$ Hamming code discussed in Example 8.5 (Chapter 8) and given by the parity-check matrix in (8.16), which we reproduce here for convenience,

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{11.4}$$

The bipartite graph of this code is depicted in Figure 11.1. The bipartite graph has 7 VNs (represented by circles), each corresponding to one of the 7 code bits, and 3 CNs (represented by squares), corresponding to the 3 parity-check equations. For convenience, we enumerate the VNs from 1 to 7 (from left to right) and assume that VN i, denoted by v_i, corresponds to code bit i, i.e., to the i-th column of the parity-check matrix. Likewise, we enumerate the CNs from 1 to 3 and we assume that CN j, denoted by c_j, corresponds to parity-check equation j, i.e., to the j-th row of the parity-check matrix. Then, for each

v_i, we draw an edge e_{ij} connecting v_i to c_j if the code bit i participates in parity-check equation j or, equivalently, if the entry h_{ji} of the parity-check matrix is a one, i.e., $h_{ji} = 1$. Following this simple rule, the bipartite graph in Figure 11.1 is obtained. The neighborhood of VN v_1 is $\mathcal{N}(\mathsf{v}_1) = \{\mathsf{c}_1, \mathsf{c}_3\}$ and the neighborhood of CN c_3 is $\mathcal{N}(\mathsf{c}_3) = \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_4, \mathsf{v}_7\}$ (the VNs of $\mathcal{N}(\mathsf{c}_1)$ are shown in gray color).

Note that a code is fully specified by both $\boldsymbol{H}$ and the corresponding bipartite graph, i.e., both representations are equivalent, and given one we can easily obtain the other. Also, observe that the values of the bits connected to the same CN must sum up to zero, since the bits participating in a parity-check equation must sum up to zero.

Clearly, a bipartite graph may contain **cycles**. A cycle is defined as a sequence of edges starting and ending in the same node that form a closed path. The number of edges of the cycle is called its length. The length of the shortest cycle is called the **girth of the graph**. As we will discuss in Section 11.5, the presence of short cycles leads in general to poor code performance. Therefore, a main design principle for LDPC codes is to design codes with large girth.

Example 11.2 For the bipartite graph in Figure 11.1, the sequence of edges defined by $\mathsf{e}_{11} - \mathsf{e}_{21} - \mathsf{e}_{22} - \mathsf{e}_{42} - \mathsf{e}_{43} - \mathsf{e}_{13}$ (shown in bold) form a cycle of length 6 starting and ending at VN v_1. This is not the shortest cycle of the graph. Indeed, the girth of the code is 4. For instance, the sequence of edges $\mathsf{e}_{11} - \mathsf{e}_{21} - \mathsf{e}_{23} - \mathsf{e}_{13}$, starting and ending at VN v_1, form a cycle of length 4.

11.2.1 LDPC Codes as a Network of Connected Repetition and Single Parity-Check Codes

Let's take a closer look at the bipartite graph of Figure 11.1. A careful inspection of the graph suggests that a VN of degree d_{v} can be interpreted as a $(d_{\mathsf{v}}, 1)$ repetition code, since the bits associated to its adjacent edges must all be equal (and this precisely corresponds to the code constraints of a repetition code!). On the other hand, since all code bits associated to the adjacent edges of a CN must sum up to zero, a CN of degree d_{c} is nothing but a $(d_{\mathsf{c}}, d_{\mathsf{c}} - 1)$ single parity-check (SPC) code. Therefore, LDPC codes can be interpreted as a network of connected repetition and SPC codes.

11.3 Degree Distributions

The following definition will be useful.

Definition 11.1 The **degree** of a node is the number of its adjacent edges.

Thus, if the i-th column of $\boldsymbol{H}$ has Hamming weight $w_{\mathsf{c},i}$, the corresponding VN in the bipartite graph, v_i, has degree $\deg(\mathsf{v}_i) = w_{\mathsf{c},i}$. Likewise, if the i-th row of $\boldsymbol{H}$ has Hamming weight $w_{\mathsf{r},i}$, the corresponding CN in the bipartite graph, c_i, has degree $\deg(\mathsf{c}_i) = w_{\mathsf{r},i}$. We will denote by $d_{\mathsf{v},i}$ the degree of the i-th VN, i.e., $d_{\mathsf{v},i} = \deg(\mathsf{v}_i)$, and by $d_{\mathsf{c},i}$ the degree of the i-th CN, i.e., $d_{\mathsf{c},i} = \deg(\mathsf{c}_i)$.

Note that for regular LDPC codes, all VNs and all CNs have the same degree, i.e., $d_{\mathsf{v},i} = d_{\mathsf{v}}\ \forall i$ and $d_{\mathsf{c},i} = d_{\mathsf{c}}\ \forall i$. Furthermore, $d_{\mathsf{v}} = w_{\mathsf{c}}$ and $d_{\mathsf{c}} = w_{\mathsf{r}}$. The code rate for a regular LDPC code can now be defined in terms of the VN and CN degrees,

$$R_{\mathsf{c}}^{\mathsf{reg}} \geq 1 - \frac{d_{\mathsf{v}}}{d_{\mathsf{c}}}, \tag{11.5}$$

with equality if $\boldsymbol{H}$ is of full rank.

The performance of LDPC codes is intimately related to their degree distribution. In particular, if decoded using low-complexity iterative decoding (described in Section 11.4 below) irregular LDPC codes perform much closer to capacity than regular LDPC codes. It is useful to specify the VN and CN degree distributions in polynomial form. In particular, the **node-perspective** VN degree distribution is defined as

$$\mathsf{\Lambda}(x) = \sum_i \mathsf{\Lambda}_i x^i, \tag{11.6}$$

where $\mathsf{\Lambda}_i$ is the probability that a VN has degree i or, equivalently, the fraction of VNs of degree i. Similarly, the node-perspective CN degree distribution is defined as

$$\mathsf{P}(x) = \sum_i \mathsf{P}_i x^i, \tag{11.7}$$

where P_i is the probability that a CN has degree i or, equivalently, the fraction of CNs of degree i. Clearly, for regular LDPC codes

$$\mathsf{\Lambda}(x) = x^{d_{\mathsf{v}}} \qquad \text{and} \qquad \mathsf{P}(x) = x^{d_{\mathsf{c}}}.$$

The average VN and CN degrees can be easily obtained from the node-perspective VN and CN degree distributions as

$$\bar{d}_{\mathsf{v}} = \sum_i i\mathsf{\Lambda}_i \qquad \text{and} \qquad \bar{d}_{\mathsf{c}} = \sum_i i\mathsf{P}_i.$$

Example 11.3 For the bipartite graph of the $(7,4)$ Hamming code of Example 11.1, drawn in Figure 11.1, the node-perspective VN and CN degree distributions are

$$\mathsf{\Lambda}(x) = \frac{3}{7}x + \frac{3}{7}x^2 + \frac{1}{7}x^3 \qquad \text{and} \qquad \mathsf{P}(x) = x^4. \tag{11.8}$$

Furthermore, $\bar{d}_{\mathsf{v}} = \frac{3}{7} + \frac{6}{7} + \frac{3}{7} = \frac{12}{7} = 1.714$ and $\bar{d}_{\mathsf{c}} = 4$.

We remark that the $(7,4)$ Hamming code is not an LDPC code, since its parity-check matrix is not sparse. However, it is very useful to introduce the concepts of bipartite graph and degree distributions.

11.4 Decoding LDPC Codes: Belief Propagation Decoding

As we discussed in Section 10.4 for turbo codes, the optimum (MAP) decoder should compute the log-APPs

$$L_{\mathsf{APP}}(u_i|\boldsymbol{y}) \triangleq \ln \frac{P_{\mathsf{APP}}(u_i = 0|\boldsymbol{y})}{P_{\mathsf{APP}}(u_i = 1|\boldsymbol{y})},$$

or, equivalently,

$$L_{\mathsf{APP}}(c_i|\boldsymbol{y}) \triangleq \ln \frac{P_{\mathsf{APP}}(c_i = 0|\boldsymbol{y})}{P_{\mathsf{APP}}(c_i = 1|\boldsymbol{y})}. \tag{11.9}$$

Unfortunately, since LDPC codes are typically very long codes (the code length is typically in the order of a few thousands of bits or even a few tens of thousands of bits), MAP decoding is not feasible.

Similar to turbo codes, LDPC codes may be decoded using a suboptimal iterative decoding algorithm based on the iterative exchange of messages along the edges of the bipartite graph, and referred to as **message**

passing decoding or **belief propagation (BP) decoding**. The idea is to interpret the LDPC code as a network of connected repetition and SPC codes, thus one may decode each repetition and SPC code optimally (using MAP decoding) and exchange extrinsic information between the component decoders iteratively. Note that this principle is very similar to the iterative turbo decoding algorithm discussed in Section 10.4. In fact, the turbo decoding algorithm can be seen as an instance of BP decoding.

In the following, we briefly describe BP decoding for LDPC codes. We start with the MAP decoding of a repetition code, since it is one of the basic building blocks of the decoder of LDPC codes.

11.4.1 MAP Decoding of Repetition Codes

Let's consider an $(n,1)$ repetition code and transmission over a memoryless channel. We denote the codeword by $\boldsymbol{c}=(c_1,c_2,\ldots,c_n)$ and the corresponding received vector by $\boldsymbol{y}=(y_1,y_2,\ldots,y_n)$. The MAP decoder computes

$$L_{\mathsf{APP}}(c_i|\boldsymbol{y}) \triangleq \ln\frac{P_{\mathsf{APP}}(c_i=0|\boldsymbol{y})}{P_{\mathsf{APP}}(c_i=1|\boldsymbol{y})}$$

for bit c_i.

Let's write first $P_{\mathsf{APP}}(c_i|\boldsymbol{y})$ as the marginalization of the joint distribution $P_{\mathsf{APP}}(c_1,\ldots,c_n|\boldsymbol{y})$ with respect to all c_j, $j=1,\ldots,n$, $j\neq i$, i.e., the marginalization with respect to all c_j except, of course, c_i,

$$P^{\mathsf{rep}}_{\mathsf{APP}}(c_i|\boldsymbol{y})=\sum_{\sim c_i}P(c_1,\ldots,c_n|\boldsymbol{y})\mathbb{I}\{c_1=c_2=\ldots=c_n\}, \tag{11.10}$$

where $\sim c_i$ is a shorthand notation indicating that the sum is over $j=1,\ldots,n$, $j\neq i$, i.e., over all c_j except c_i, $\mathbb{I}\{c_1=c_2=\ldots=c_n\}$ is the indicator function that reveals the constraints of a repetition code (all bits must be equal), and we use the superindex rep to stress the fact that these APPs are for a repetition code.

We can expand (11.10) as

$$\begin{aligned}
P^{\mathsf{rep}}_{\mathsf{APP}}(c_i|\boldsymbol{y}) &= \sum_{\sim c_i}P(c_1,\ldots,c_n|\boldsymbol{y})\mathbb{I}\{c_1=c_2=\ldots=c_n\}\\
&\overset{(a)}{=} \sum_{\sim c_i}\frac{P(\boldsymbol{y}|\boldsymbol{c})P(\boldsymbol{c})}{P(\boldsymbol{y})}\prod_{j\neq i}\mathbb{I}\{c_j=c_i\}\\
&\overset{(b)}{=} \frac{1}{P(\boldsymbol{y})}\sum_{\sim c_i}\prod_{j=1}^{n}P(y_j|c_j)P(c_j)\prod_{j\neq i}\mathbb{I}\{c_j=c_i\}\\
&= \frac{1}{P(\boldsymbol{y})}P(y_i|c_i)P(c_i)\sum_{\sim c_i}\prod_{j\neq i}P(y_j|c_j)P(c_j)\mathbb{I}\{c_j=c_i\}\\
&\overset{(c)}{=} \frac{1}{P(\boldsymbol{y})}P(y_i|c_i)P(c_i)\prod_{j\neq i}P(y_j|c_i)P(c_j=c_i),
\end{aligned}$$

where in (a) we simply used Bayes' rule and factorized $\mathbb{I}\{c_1=c_2=\ldots=c_n\}$ into the product of indicator functions, in (b) we used the fact that $P(\boldsymbol{y}|\boldsymbol{c})P(\boldsymbol{c})=\prod_{j=1}^{n}P(y_j|c_j)P(c_j)$ because the channel is memoryless, and finally (c) follows from $P(c_j)\mathbb{I}\{c_j=c_i\}=P(c_j=c_i)$.

The log-APPs for the repetition code can then be written as

$$
\begin{aligned}
L_{\mathsf{APP}}^{\mathsf{rep}}(c_i|\boldsymbol{y}) &\triangleq \ln \frac{P_{\mathsf{APP}}^{\mathsf{rep}}(c_i=0|\boldsymbol{y})}{P_{\mathsf{APP}}^{\mathsf{rep}}(c_i=1|\boldsymbol{y})}\\
&= \ln \frac{P(y_i|c_i=0)P(c_i=0)\prod_{j\neq i}P(y_j|c_j=0)P(c_j=0)}{P(y_i|c_i=1)P(c_i=1)\prod_{j\neq i}P(y_j|c_j=1)P(c_j=1)}\\
&= \ln \frac{P(y_i|c_i=0)}{P(y_i|c_i=1)} + \ln \frac{P(c_i=0)}{P(c_i=1)} + \sum_{j\neq i} \ln \frac{P(y_j|c_j=0)}{P(y_j|c_j=1)} + \sum_{j\neq i} \ln \frac{P(c_j=0)}{P(c_j=1)},
\end{aligned}
\tag{11.11}
$$

where

$$
L_{\mathsf{A}}^{\mathsf{rep}}(c_i) = \ln \frac{P(c_i=0)}{P(c_i=1)}
$$

is the a priori information on the code bit c_i, $i=1,\ldots,n$, of the repetition code and

$$
L_{\mathsf{ch}}(c_i) = \ln \frac{P(y_i|c_i=0)}{P(y_i|c_i=1)}
$$

is the channel LLR for bit c_i.

Therefore, (11.11) can be rewritten as

$$
\begin{aligned}
L_{\mathsf{APP}}^{\mathsf{rep}}(c_i|\boldsymbol{y}) &= L_{\mathsf{ch}}(c_i) + L_{\mathsf{A}}^{\mathsf{rep}}(c_i) + \underbrace{\sum_{j\neq i}(L_{\mathsf{ch}}(c_j) + L_{\mathsf{A}}^{\mathsf{rep}}(c_j))}_{\text{extrinsic information}}\\
&= L_{\mathsf{ch}}(c_i) + L_{\mathsf{A}}^{\mathsf{rep}}(c_i) + L_{\mathsf{E}}^{\mathsf{rep}}(c_i),
\end{aligned}
$$

where

$$
L_{\mathsf{E}}^{\mathsf{rep}}(c_i) = \sum_{j\neq i}(L_{\mathsf{ch}}(c_j) + L_{\mathsf{A}}^{\mathsf{rep}}(c_j)) \tag{11.12}
$$

is the extrinsic information on bit c_i.

In this course, we will not compute the APPs and the extrinsic information for an SPC code, since the computation is more involved. However, the interested reader is referred to [9, Sec. 5.4.3].

11.4.2 Belief Propagation Decoding

We can now describe the iterative (BP) decoding of LDPC codes to compute (11.9) approximately.

Consider a generic VN, v, and a generic CN, c, depicted in Figure 11.2. For notational simplicity, we denote the channel LLR for code bit c_i, i.e., the channel LLR associated to VN v_i by

$$
L_i \triangleq L_{\mathsf{ch}}(c_i) \triangleq \ln \frac{P(y_i|c_i=0)}{P(y_i|c_i=1)}.
$$

The VNs and the CNs exchange extrinsic information iteratively along the edges of the bipartite graph. Denote by $L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i}$ the extrinsic information that CN c_j sends to VN v_i (obviously there must be an edge connecting CN c_j to VN v_i) and by $L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j}$ the extrinsic information that VN v_i sends to CN c_j over the corresponding edge in the bipartite graph. The extrinsic information $L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i}$ will be used as a priori information by the VN v_i when decoding the corresponding repetition code, i.e., we can write

$$
L_{\mathsf{A}}^{\mathsf{c}_j\to\mathsf{v}_i} = L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i}.
$$

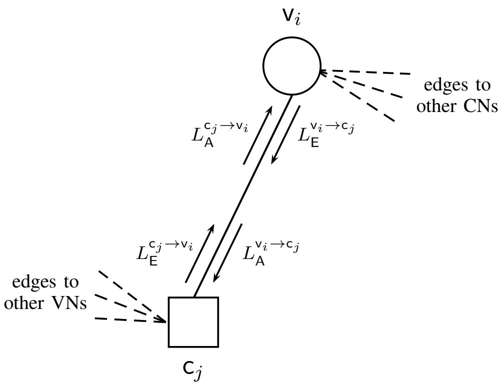

v_i
edges to other CNs
$L_{\mathsf{A}}^{\mathsf{c}_j\to\mathsf{v}_i}$
$L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j}$
$L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i}$
$L_{\mathsf{A}}^{\mathsf{v}_i\to\mathsf{c}_j}$
edges to other VNs
c_j

Figure 11.2: Belief propagation decoding of LDPC codes. The VNs and CNs iteratively exchange extrinsic information along the edges of the bipartite graph.

Similarly, we can write

$$L_{\mathsf{A}}^{\mathsf{v}_i\to\mathsf{c}_j} = L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j}$$

to indicate the a priori information that CN c_j receives from VN v_i, which is the extrinsic information $L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j}$ computed by VN c_j.

The BP decoding proceeds as follows.

1. At iteration ℓ:

   (a) Decode all VNs (i.e., all repetition codes) and pass extrinsic information to all CNs. The VN SISO decoders receive at their input the channel LLRs L_i and extrinsic information from the neighboring CNs, which is used as a priori information. In particular, VN v_i computes the extrinsic information $L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j}$ to be sent to node CN c_j according to (11.12),

   $$L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j,(\ell)} = L_i + \sum_{\mathsf{c}_k\in\mathcal{N}(\mathsf{v}_i)/\mathsf{c}_j} L_{\mathsf{A}}^{\mathsf{c}_k\to\mathsf{v}_i,(\ell)}, \tag{11.13}$$

   where $L_{\mathsf{A}}^{\mathsf{c}_k\to\mathsf{v}_i,(\ell)}$ is the extrinsic information generated by CN c_k in the previous iteration, i.e.,

   $$L_{\mathsf{A}}^{\mathsf{c}_k\to\mathsf{v}_i,(\ell)} = L_{\mathsf{E}}^{\mathsf{c}_k\to\mathsf{v}_i,(\ell-1)}.$$

   Note that at the first iteration the CNs do not provide any a priori information, i.e., $L_{\mathsf{E}}^{\mathsf{c}_k\to\mathsf{v}_i,(0)} = 0$, and (11.13) reduces to

   $$L_{\mathsf{E}}^{\mathsf{v}_i\to\mathsf{c}_j,(1)} = L_i.$$

   (b) Decode all CNs (i.e., all SPC codes) and pass extrinsic information to all VNs. The decoding rule is a bit more complicated than for the VNs (see [9, Sec. 5.4.3]). In particular, CN c_i computes the extrinsic information $L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i}$ to be sent to node VN v_i as

   $$L_{\mathsf{E}}^{\mathsf{c}_j\to\mathsf{v}_i,(\ell)} = 2\tanh^{-1}\left[\prod_{\mathsf{v}_k\in\mathcal{N}(\mathsf{c}_j)/\mathsf{v}_i} \tanh\left(\frac{L_{\mathsf{A}}^{\mathsf{v}_k\to\mathsf{c}_j,(\ell)}}{2}\right)\right],$$

   where $L_{\mathsf{A}}^{\mathsf{v}_k\to\mathsf{c}_j,(\ell)}$ is the extrinsic information generated by VN v_k at iteration ℓ, i.e.,

   $$L_{\mathsf{A}}^{\mathsf{v}_k\to\mathsf{c}_j,(\ell)} = L_{\mathsf{E}}^{\mathsf{v}_k\to\mathsf{c}_j,(\ell)}.$$

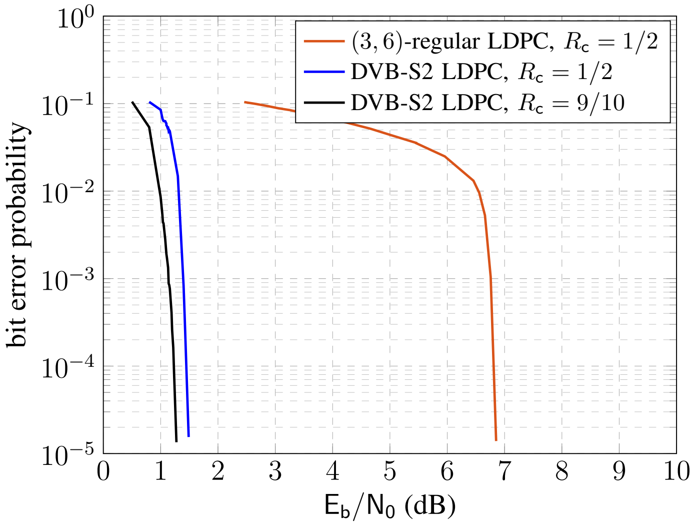


Figure 11.3: BER performance of the $(3, 6)$ regular LDPC code, with code length $n = 100000$, and of the DVB-S2 code, with $n = 64800$.

2. Repeat Step 1 until a maximum number of iterations ℓ_{max} is reached. Then make decisions on the code bits c_i according to

$$\hat{c}_i = \begin{cases} 1 & \text{if } L_{\mathsf{APP}}^{(\ell_{\mathsf{max}})}(c_i) < 0 \\ 0 & \text{if } L_{\mathsf{APP}}^{(\ell_{\mathsf{max}})}(c_i) > 0 \end{cases},$$

where

$$L_{\mathsf{APP}}^{(\ell_{\mathsf{max}})}(c_i) = L_i + \sum_{\mathsf{c}_k \in \mathcal{N}(\mathsf{v}_i)} L_{\mathsf{A}}^{\mathsf{c}_k \to \mathsf{v}_i, (\ell_{\mathsf{max}})}.$$

Remark 11.1 The BP decoding algorithm makes an optimal decision, i.e., it computes exactly the log-APPs (11.9) if the code graph contains no cycles. On the contrary, if the graph contains cycles (which is always the case), then the BP algorithm is suboptimal. Nonetheless, it provides excellent performance.

11.5 Performance of Low-Density Parity-Check Codes

As turbo codes, LDPC codes achieve excellent performance very close to the Shannon limit. Furthermore, qualitatively, the performance of LDPC codes is very similar to that of turbo codes: it is characterized by a waterfall region and an error floor region. When the block length goes to infinity, the performance of LDPC codes (and of turbo codes as well) is characterized by a so-called threshold effect, i.e., there is a value of $\mathsf{E_b}/\mathsf{N_0}$ below which the BER goes to zero.

Under BP decoding, the performance depends greatly on the degree distributions $\Lambda(x)$ and $\mathsf{P}(x)$. Typically, for a given channel, $\Lambda(x)$ and $\mathsf{P}(x)$ must be optimized to yield the best possible decoding threshold. The resulting optimal distributions are in general irregular. In other words, irregular LDPC codes perform closer to capacity than regular LDPC codes. Then, for a given code length, one designs a particular LDPC code with the optimal degree distributions $\Lambda(x)$ and $\mathsf{P}(x)$ found from the optimization. To yield low error floors, short cycles in the graph must be avoided, in particular cycles of length 4. Therefore, given the degree distributions $\Lambda(x)$ and $\mathsf{P}(x)$ one should construct a code with large girth.

In Figure 11.3, we plot the BER performance of a $(d_{\mathsf{v}} = 3, d_{\mathsf{c}} = 6)$ regular LDPC code (note that, assuming that the parity-check matrix is of full rank, the code rate is $R_{\mathsf{c}} = 1/2$, see (11.5)) and of the irregular LDPC code used in the digital video broadcasting (DVB-S2) standard for rates $R_{\mathsf{c}} = 1/2$ and $R_{\mathsf{c}} = 9/10$. The degree distributions of the $R_{\mathsf{c}} = 1/2$ DVB-S2 LDPC code are $\mathsf{\Lambda}(x) = \frac{1}{64800}x + \frac{32399}{64800}x^2 + 0.3x^3 + 0.2x^8$ and $\mathsf{P}(x) = \frac{1}{32400}x^6 + \frac{32399}{32400}x^7$. Note that for $R_{\mathsf{c}} = 1/2$, the DVB-S2 code performs closer to the Shannon limit than the $(3, 6)$ regular LDPC code, despite a shorter code length. Observe also that the DVB-S2 LDPC code performs relatively close to capacity for $R_{\mathsf{c}} = 1/2$ (for BPSK transmission the Shannon limit is $\mathsf{E_b}/\mathsf{N_0} = 0.187$ dB), but more than 2 dB away from capacity for $R_{\mathsf{c}} = 9/10$ (the Shannon limit is $\mathsf{E_b}/\mathsf{N_0} = 3.199$ dB). However, it should be noted the DVB-S2 code is a poorly performing LDPC code, especially for high rates! Nowadays there exist LDPC codes that perform very close to capacity also for very high rates.

Acknowledgements

The author would like to thank Arni Alfredsson, Andreas Buchberger, and Christian Häger for proof-reading several versions of the lecture notes, and for valuable comments and corrections. Thanks also to Christian and Arni for their help in drawing several of the figures.

Bibliography

[1] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[2] M. A. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[3] G. Liva, “Graph-based analysis and optimization of contention resolution diversity slotted ALOHA,” *IEEE Trans. Commun.*, vol. 59, no. 2, pp. 477–487, Feb. 2011.

[4] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “A unified coding framework for distributed computing with straggling servers,” in *Proc. IEEE Globecom Work. (GC Wkshps)*, Washington, DC, USA, Dec. 2016.

[5] ——, “Coding for distributed fog computing,” *IEEE Commun. Magazine*, vol. 55, no. 4, pp. 34–40, Apr. 2017.

[6] D. Pearson, “High-speed QKD reconciliation using forward error correction,” in *AIP Int. Conf. Quantum Commun., Measurement and Computing*, vol. 734, no. 1, Glasgow,UK, 2004, pp. 299–302.

[7] R. J. McEliece, “A public-key cryptosystem based on algebraic coding theory,” *Deep Space Network Progress Report*, vol. 44, pp. 114–116, Jan. 1978.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proc. IEEE Int. Conf. Communications (ICC)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[9] W. E. Ryan and S. Lin, *Channel codes. Classical and modern.* Cambridge: Cambridge University Press, 2009.

[10] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, “Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding,” *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 909–926, May 1998.

[11] C. Berrou, A. Graell i Amat, Y. Ould-Cheikh-Mouhamedou, and Y. Saouter, “Improving the distance properties of turbo codes using a third component code: 3d turbo codes,” *IEEE Trans. Commun.*, vol. 57, no. 9, pp. 2505–2509, Sep. 2009.

[12] R. M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep 1981.