Introduction to Communication Theory

In this course we will cover different aspects of the problem of transmitting data through a channel, which is the main object of study in Communication Theory.

There are three main reasons why one would like to transmit encoded data instead of raw data:

1) efficiency (Information Theory)

2) robustness (Coding Theory)

3) secrecy (Cryptography)

The first two are closely related: we aim to compress our data as much as possible and to protect it from possible errors that may occur. While in real-life applications both efficiency and robustness are desirable simultaneously, here we will consider them separately. In the first part of the course we will cover the part concerning Information theory and, in the second part, we will focus on the study of Coding Theory.

Historical remark: Information and Coding Theory were both born in the late 1940s:

1948: Claude E. Shannon wrote *"A Mathematical Theory of Communication"*. In it, Shannon proved the Noiseless and Noisy Coding Theorems. We will cover these two theorems in the course. First appearance of the word *bit* to refer to BInary digiT. Shannon is considered to be the father of these two areas.

1950: Richard W. Hamming wrote *"Error Detecting and Error Correcting Codes"*. In it, Hamming constructed the first Error-Correcting code. We will study Hamming codes in the second part of the course.

Further topics. A timeline of advances in Communication Theory can be found in:

`https://en.wikipedia.org/wiki/Timeline_of_information_theory`

Information Theory: The goal of Information Theory is to maximize the amount of information in the transmitted data.

The following example illustrates the purpose of Information Theory. Imagine that, at each second, we are required to inform about the weather in the Death Valley (California, US) through a noiseless channel. The possibilities are:

sunny, cloudy, rainy, stormy, foggy, snowy, lightning and hail.

Since $8 = 2^3$, we can assign to each type of weather a unique identifier using a 3-bit string: sunny$\to$ 000, cloudy$\to$ 001, ..., hail$\to$ 111. At each second, we can send the identifier that corresponds to current weather. Thus, at every second we will send 3 bits.

According to past records, on average, there are 291 sunny days per year in the Death Valley. Thus, the probability of sunny weather is $291/365 \approx 0.8$. Can we use this information to transmit in a more efficient way? Intuitively speaking, since it is sunny most of the time it might be convenient to give it a shorter identifier, even if the other identifiers become longer. Suppose that the following identifiers are assigned: sunny$\to$ 0, cloudy$\to$ 1001, ..., hail$\to$ 1111. In the worst case, at every second we will send 4 bits, but, on average, we will send $0.8 \cdot 1 + 0.2 \cdot 4 = 1.6$ bits.

The goal of the first part of the course will be to design efficient/optimal encodings, given the frequencies of the source.

Some real-world examples of encoding for efficiency:

- ASCII code.
- MORSE code.
- ZIP data compression.
- Lossless Image compression format, such as PNG.
- Lossy Image compression format, such as JPEG.

Coding Theory: The goal of Coding Theory is to add redundancy on the data transmitted through a noisy channel in order to protect it from possible errors. The main tools we will use come from Combinatorics and from Linear Algebra.

The English vocabulary can be understood as an error-correcting code that we use to communicate. There are roughly 200000 words in the Oxford English Dictionary and 26 letters in the alphabet. Since $26^4 > 200000$, if we only care about efficiency, we could assign a 4-letter word to each concept in the dictionary. However, the communication would be complicated (some words would be unpronounceable and hard to remember) and misunderstandings would be frequent. Instead, we use longer words that reduce the number of misunderstandings.

For example, suppose we receive the following message:

Welfome to Cojbinatomics anp Comtunicarion Thefry

Here our brain works as a decoder, mapping each string to the "most likely" word. For instance, it is natural to decode the previous message as *Welcome to Combinatorics and Communication Theory*. Another example of a decoder is the autocorrector in your mobile's keyboard. If a word is not in the dictionary, the autocorrector will change it to the closest one that is there.

Consider now the following message:

Goor dight

Here it is not that obvious which is the original message; some candidates are *Good night*, *Poor light* and *Good digit*. The robustness of the English as an error-correcting code highly depends on the words we use: words like *communication* can be decoded even if several errors occur, while words like *poor* may not be recoverable even if only one error occurs.

The goal of the second part of the course will be to design binary languages (called binary codes) that allow us to detect/correct many errors while minimising the amount of redundant information that is sent.

Some real-world examples of encoding for robustness:

- CDs and DVDs
- ISBN numbers
- QR codes
- Satellite Communications
- Bank Transactions

Chapter 1

Introduction to Codes

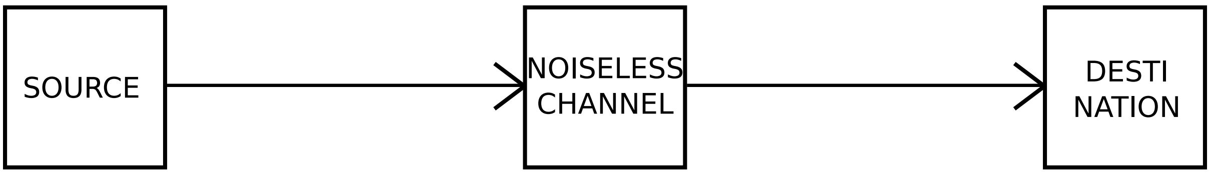
In this first chapter we introduce codes and their basic properties.

For $n \in \mathbb{N}$, define $[n] := \{1, 2, \ldots, n\}$.

The *channel alphabet* is a finite set of symbols Σ. For every $n \geq 0$, we denote by Σ^n the set of strings of length n over Σ. We also denote by $\Sigma^* := \cup_{n \geq 0} \Sigma^n$ the set of strings of any length over Σ. Unless we state otherwise, during this course we will only consider $\Sigma = \{0, 1\}$ - in this case the channel is called *binary* and the symbols are called *bits*.

Definition 1.1 (Code, codeword)**.** A (binary) *code* C is a finite subset of $\{0, 1\}^*$. The elements of C are called *codewords* and denoted by $c_1, c_2, \ldots$. The *length* of $c \in C$ is the length of the string that represents it and it is denoted by $|c|$.

Simplest Communication Scheme: In this chapter we will consider the following communication scheme:



Source and Destination agree to use a code C. Source sends a sequence of codewords to Destination through a noiseless channel, bit by bit. The goal of Destination is to split the stream of bits into the sequence codewords that have been sent.

1.1 Properties of Codes

Definition 1.2 (Uniquely decipherable)**.** A code C is *uniquely decipherable* if for every $n \geq 1$ and every sequence $x_1 x_2 \ldots x_n \in \{0, 1\}^n$ there exists at most one sequence of codewords $c_1, c_2, \ldots, c_m \in C$ such that $x_1 x_2 \ldots x_n = c_1 c_2 \ldots c_m$.

Equivalently, C is uniquely decodable if and only if, for every $c_1, \ldots, c_{m_1}, d_1, \ldots, d_{m_2} \in C$ with

$$c_1 \ldots c_{m_1} = d_1 \ldots d_{m_2} \ ,$$

we have $m_1 = m_2$ and $c_i = d_i$ for every $i \in [m_1]$.

Observe that we do not require every sequence in $\{0, 1\}^*$ to be decodable, but only that there is *at most one* way to decode it.

Example 1.3. The code $C_1 = \{010, 10, 101\}$ is not uniquely decipherable. Assume that the string 101010 is received. This could correspond to the message composed either by c_3c_1 or by $c_2c_2c_2$.

The code $C_2 = \{0, 011, 101\}$ is uniquely decipherable. Let $x_1x_2 \dots x_n \in \{0,1\}^n$ be a string of bits that corresponds to at least one sequence of codewords in C_2. It suffices to uniquely identify the first codeword in the string. If so, we can process the remainder substring as a shorter string. We do it by case analysis:

- if $x_1 = 1$, the first codeword is $c_3 = 101$,
- if $x_1 = 0$, then
    - if $x_2 = 0$ the first codeword is $c_1 = 0$.
    - if $x_2 = 1$, then
        - if $x_3 = 0$, the first codeword is $c_1 = 0$ (and the second one, $c_3 = 101$).
        - if $x_3 = 1$, the first codeword is $c_2 = 011$.

For instance, the string 00110110101 corresponds to a unique sequence of codewords: $0|011|011|0|101 = c_1c_2c_2c_1c_3$. Observe that there are some strings in $\{0,1\}^*$, like 0100111 that do not correspond to any sequence of codewords.

Definition 1.4 (Instantaneous)**.** A code C is *instantaneous* if every codeword $c = x_1 \dots x_\ell \in C$ can be identified as soon as its last bit x_ℓ is received.

Example 1.5. The code $C_1 = \{0, 10, 110, 111\}$ is instantaneous. However, the code $C_2 = \{0, 011, 101\}$ defined in Example 1.3 is not instantaneous: the codeword 0 cannot be identified as soon as it is received, since the string 0 could correspond either to the codeword $c_1 = 0$ or to the initial part of the codeword $c_2 = 011$.

A string $x \in \{0,1\}^*$ is a *prefix* of $x' \in \{0,1\}^*$ if there exists a non-empty string $y \in \{0,1\}^*$ such that $x' = xy$.

Definition 1.6 (Prefix-free)**.** A code C is *prefix-free* if no codeword is the prefix of another one. In other words, if $c = x_1 \dots x_\ell \in C$, then $x_1 \dots x_i \notin C$ for every $i \in [\ell - 1]$.

Example 1.7. In Example 1.5, the code C_1 is prefix-free, while the code C_2 is not: $c_2 = 10$ is a prefix of $c_3 = 101$.

Examples 1.5 and 1.7 motivate the following result.

Theorem 1.8. *A code C is instantaneous if and only if it is prefix-free.*

Proof. Let C be a prefix-free code and let $x_1x_2 \dots \in \{0,1\}^*$. Suppose that the codeword $c = x_1x_2 \dots x_\ell \in C$ is transmitted. Since the code is prefix-free, there is no other codeword of the form $c' = x_1x_2 \dots x_\ell y$, for a non-empty $y \in \{0,1\}^*$. Thus, as soon as we receive the last bit x_ℓ, we can immediately identify $x_1x_2 \dots x_\ell$ with c. So C is instantaneous.

Assume now that C is instantaneous, but, for the sake of contradiction, there exist two codewords c_1, c_2 such that c_1 is a prefix of c_2; that is, $c_1 = x_1x_2 \dots x_\ell$ and $c_2 = x_1x_2 \dots x_\ell y$, for a non-empty $y \in \{0,1\}^*$. If we receive the string of bits $x_1x_2 \dots x_\ell \in \{0,1\}^*$ then we have two options: identify it as the codeword c_1 or wait for more bits to be transmitted, as it could be the initial part of c_2. So C is not instantaneous, leading to a contradiction. $\square$

Theorem 1.9. *If a code C is an instantaneous code, then it is uniquely decipherable.*

Proof. Let C be an instantaneous code. Consider a string in $\{0,1\}^*$. Since C is instantaneous, every time we receive the last bit of a codeword we can unambiguously identify it. Thus, there is at most one way to decode the string into codewords of C and C is uniquely decipherable. $\square$

Example 1.10. Not all the uniquely decipherable codes are instantaneous. Consider again $C_2 = \{0, 011, 101\}$. In Exercises 1.3 and 1.5, we have proved that it is uniquely decipherable and that it is neither instantaneous nor prefix-free.

The following summarises the results of this section:

$$\text{Prefix-free} \overset{\text{Thm 1.8}}{\Longleftrightarrow} \text{Instantaneous} \underset{\text{Exm 1.10}}{\overset{\text{Thm 1.9}}{\substack{\Longrightarrow \\ \not\Longleftarrow}}} \text{Uniquely decipherable}$$

Example 1.11. The *reversal* code of C is the code C^r obtained by taking the codewords of C in the reverse order. For instance, if $C = \{10, 001, 101\}$, then $C^r = \{01, 100, 101\}$.

- The reversal code of a prefix-free code is not always prefix-free. For example, if $C = \{0, 10\}$, then C is prefix-free but its reversal $\{0, 01\}$ is not.

- The reversal code of a uniquely decipherable code is uniquely decipherable. Suppose it is not. There there exists a string $x_1 \dots x_n \in \{0,1\}^*$ that can be decoded in two different ways using codewords from C^r. Say that $x_1 \dots x_n = c_1^r \dots c_m^r$ and $x_1 \dots x_n = d_1^r \dots d_t^r$ with $c_i^r, d_j^r \in C^r$. Consider now the reverse string $x_n \dots x_1 \in \{0,1\}^*$. Note that, for every $i \in [m]$, the reverse of c_i^r, denoted by c_i, is a codeword of C. In a similar way, for every $j \in [t]$, the reverse of d_j^r, denoted by d_j, is a codeword of C. Then, $x_\ell \dots x_1 = c_m \dots c_1$ and $x_\ell \dots x_1 = d_t \dots d_1$ and the reverse string can be decoded in two different ways using codewords of C. This is a contradiction with the fact that C is uniquely decipherable.

1.2 Binary Trees and prefix-free Codes

We will establish a bijection between binary prefix-free codes and edge-labelled binary trees. We start with some definitions

A *tree* T is a connected graph with no cycles. A *rooted tree* T is a tree together with a distinguished vertex r, which is called the *root* of T. Recall that, for every vertex v different from r, there is a unique path that connects v to r in T. For a vertex v of T different from r, the *parent* of v is the neighbour of v that lies in the unique path between v and r. The other neighbours of v are its *children*. Vertices that have no children are the *leaves* of the tree. A *binary tree* is a rooted tree where every vertex has at most two children. We consider binary trees in which the edges connecting to the children of a vertex are assigned distinct labels from $\{0,1\}$. It is useful to think about the edge-label as 0 for the *left* child and 1 for the *right* one.

Figure 1.1 shows an example of an edge-labelled binary tree: We start by defining the notion of code associated to an edge-labelled binary tree.

Definition 1.12 (Associated code of a tree)**.** The *associated code* of an edge-labelled binary tree T with root r is constructed by creating a codeword for each leaf v of T as follows: concatenate all the edge labels in the unique path from r to v.

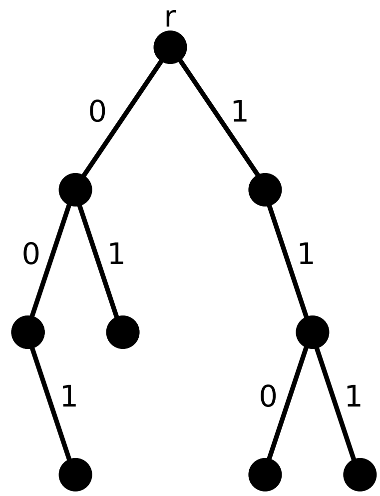


Figure 1.1: A binary edge-labelled tree

Example 1.13. Consider the edge-labelled binary tree in Figure 1.1. Its associated code is $\{001, 01, 110, 111\}$.

Lemma 1.14. *The associated code of an edge-labelled binary tree with m leaves is a prefix-free code of size m.*

Proof. Let T be a binary tree and let C be its associated code. Clearly, if T has m leaves, C has size m. For the sake of contradiction, suppose that C is not prefix-free. Therefore, there exist $c_1, c_2 \in C$ such that c_1 is a prefix of c_2. If $c_1 = x_1 \dots x_{\ell_1}$ and $c_2 = y_1 \dots y_{\ell_2}$, then we have $\ell_2 > \ell_1 \geq 1$ and $x_i = y_i$ for every $i \in [\ell_1]$. Let P_1 be the path obtained by starting at the root r of T and at the i-th step following the edge labelled by x_i, for $i \in [\ell_1]$. Analogously, let P_2 be the path obtained by starting at the root r of T and at the ith step following the edge labelled by y_i, for $i \in [\ell_2]$. Note that each such path contains exactly one leaf in T. Let v_1 be the leaf in P_1 and v_2 the one in P_2. By construction and since $x_i = y_i$ for $i \in [\ell_1]$, the path P_1 is included in P_2, therefore v_1 is also a leaf of P_2. This implies that $v_1 = v_2$, and thus $c_1 = c_2$. This is a contradiction since $|c_1| = \ell_1 < \ell_2 = |c_2|$. □

Definition 1.15 (Associated tree of a prefix-free code)**.** The *associated tree* of a prefix-free code C is an edge-labelled binary tree constructed as follows. For each codeword $c = x_1 \dots x_\ell \in C$, construct a path P that starts at the root r and, for every $i \in [\ell]$, at the i-th step it creates/follows an edge with label x_i.

Lemma 1.16. *The associated tree of a prefix-free code of size m has m leaves.*

Proof. Let C be a prefix-free code and T its associated tree. It is clear that it has at most m leaves, since each path contains at most one leaf and there are as many paths as codewords.

For the sake of contradiction, suppose that T has at most $m-1$ leaves. Let $v_1, \dots, v_m$ be the endpoints of the paths $P_1, \dots, P_m$ corresponding to the codewords $c_1, \dots, c_m$. We split into two cases

1) $v_1, \dots, v_m$ are leaves: by the pigeonhole principle there exists $i, j \in [m]$ with $i \neq j$ such that $v_i = v_j$. By the construction of the associated tree, this implies that $P_i = P_j$. So, $c_i = c_j$, a contradiction.

2) there exists $i \in [m]$ such that v_i is not a leaf: let v_j be a leaf in the subtree of T that is rooted at v_i. Since v_i is not a leaf, $v_j \neq v_i$. This implies that $P_i \subsetneq P_j$. So, if $c_i = x_1 \dots x_{\ell_1}$ and $c_j = y_1 \dots y_{\ell_j}$, we have that $\ell_j > \ell_i \geq 1$ and for every $k \in [\ell_i]$, $x_k = y_k$. So $y = y_{\ell_i+1} \dots y_{\ell_j}$ is a non-empty string with $c_j = c_i y$. Thus, c_i is a prefix of c_j, a contradiction.

□

We can now prove the main result of this section.

Proposition 1.17. *There is a one-to-one correspondence between prefix-free codes of size m and edge-labelled binary trees with m leaves.*

Proof. From Lemmas 1.14 and 1.16, we know we can associate a prefix-free code of size m to any binary edge-labelled tree with m leaves, and vice versa. Moreover, notice that *distinct* prefix-free codes C, C' of size m give rise to *distinct* associated trees with m leaves (and vice versa). This establishes the one-to-one correspondence. □

1.3 McMillan's and Kraft's Theorems

The goal of this section is to show that prefix-free codes cannot contain too many short codewords.

Theorem 1.18 (McMillan, 1956)**.** *Let $C = \{c_1, \dots, c_m\}$ be a prefix-free code and let $\ell_i := |c_i|$. Then*

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

Proof. Let C be a prefix-free code and let $\ell := \max_{i \in [m]}\{\ell_i\}$ be the maximum length of a codeword in C. Let T_ℓ denote the edge-labelled complete binary tree of depth ℓ; that is, the binary tree where all leaves are at distance ℓ from the root and every vertex which is not a leaf has exactly two children, the left one with label 0 and the right one with label 1.

Consider the tree T associated to C, which is contained in T_ℓ. By Lemma 1.16, T has m leaves $v_1, \dots, v_m$ corresponding to the codewords $c_1, \dots, c_m \in C$. As an example, if $C = \{000, 001, 011, 1\}$, then $\ell = 3$ and T is as in Figure 1.2.

Let V_i be the set of leaves of T_ℓ that belong to the sub-tree that is rooted at v_i. Since c_i has length ℓ_i, then v_i is at distance ℓ_i from the root and we have that $|V_i| = 2^{\ell-\ell_i}$. For instance, in the example displayed in Figure 1.2, we have $|V_1| = 1$, $|V_2| = 1$, $|V_3| = 1$ and $|V_4| = 4$. Also, since $v_1, \dots, v_m$ are leaves of T, we have $V_i \cap V_j = \emptyset$, for $i \neq j$, which implies $\sum_{i=1}^m |V_i| = |\cup_{i=1}^m V_i|$. As T_ℓ has 2^ℓ leaves, we also have $|\cup_{i=1}^m V_i| \leq 2^\ell$. We conclude that

$$\sum_{i=1}^{m} 2^{-\ell_i} = 2^{-\ell} \sum_{i=1}^{m} 2^{\ell-\ell_i} = 2^{-\ell} \sum_{i=1}^{m} |V_i| = 2^{-\ell} |\cup_{i=1}^m V_i| \leq 1 \ .$$

□

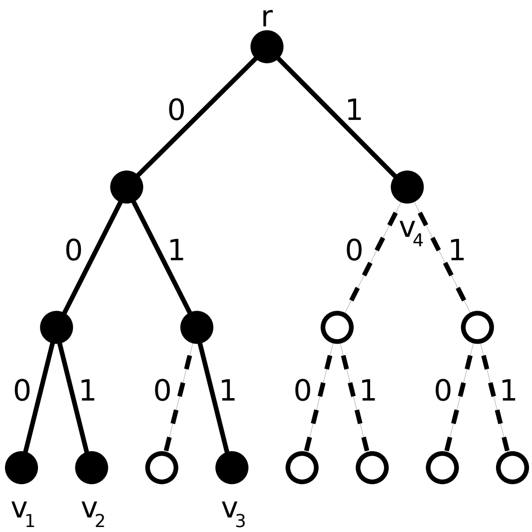


Figure 1.2: T embedded in T_3

Remark. The upper bound in McMillan's theorem is best possible. That is, there exist prefix-free binary codes for which $\sum_{i=1}^{m} 2^{-\ell_i} = 1$. For instance, fix $n \geq 1$ and consider the code C that contains all binary strings of length n.

Example 1.19. There is no prefix-free code with codewords of length $\ell_1 = \ell_2 = 2$, $\ell_3 = \ell_4 = \ell_5 = 3$ and $\ell_6 = \ell_7 = \ell_8 = 4$. Suppose C is such a prefix-free code, then

$$\sum_{i=1}^{8} 2^{-\ell_i} = 2 \cdot 2^{-2} + 3 \cdot 2^{-3} + 3 \cdot 2^{-4} = \frac{17}{16} > 1 \ ,$$

a contradiction to McMillan's theorem.

Further topics. Theorem 1.18 is not the original theorem of McMillan. Kraft proved Theorems 1.18 and 1.20 in his MSci thesis (1949). In 1956, McMillan extended Theorem 1.18 to uniquely decipherable codes, which is a wider class of codes than the class of prefix-free codes (see Theorem 1.9 and Example 1.10). Here you can find the original McMillan's theorem.

In fact, given a collection of lengths that satisfies the inequality in Theorem 1.18, one can construct a prefix-free code with the desired codeword lengths.

Theorem 1.20 (Kraft, 1949)**.** *Let $\ell_1, \ldots, \ell_m$ be a collection of positive integers that satisfy $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$. Then there is a prefix-free code whose codewords have lengths $\ell_1, \ldots, \ell_m$.*

Proof. Let a_j be the number of indices $i \in [m]$ such that $\ell_i = j$; that is $a_j := |\{i : \ell_i = j\}|$, and let $\ell := \max_{i \in [m]}\{\ell_i\}$.

We first prove that for any $j \in [\ell]$, one has

$$a_j \leq 2^j - a_1 2^{j-1} - \ldots - a_{j-1} 2 \ . \tag{1.1}$$

Using the hypothesis of the theorem, we can write

$$\sum_{k=1}^{j} a_k 2^{-k} \leq \sum_{k=1}^{\ell} a_k 2^{-k} = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1 \ .$$

We can multiply both sides by 2^j and obtain

$$\sum_{k=1}^{j} a_k 2^{j-k} \leq 2^j \ .$$

By isolating a_j in the previous inequality, we obtain (1.1).

We proceed to construct a binary code sequentially: for every $j \in [\ell]$, at the j-th step and having chosen a_k codewords of length k for every $k \in [j-1]$, we will choose a_j words of length j. We will do it in such a way that the code obtained is prefix-free

For $k \in [j-1]$, any word of length k is the prefix of 2^{j-k} words of length j. Hence, there are at most $a_1 2^{j-1} + a_2 2^{j-2} + \cdots + a_{j-1} 2$ words of length j that cannot be used if we want the code to be prefix-free. In other words, there are at least $2^j - a_1 2^{j-1} - a_2 2^{j-2} - \cdots - a_{j-1} 2$ words of length j available. By (1.1), $a_j \leq 2^j - a_1 2^{j-1} - a_2 2^{j-2} - \cdots - a_{j-1} 2$, which means that we can select a_j codewords of length j and add them to the code we are constructing while preserving the prefix-free property. □

Example 1.21. Let $\ell_1 = \ell_2 = 2$, $\ell_3 = \ell_4 = 3$ and $\ell_5 = \ell_6 = 4$. Since

$$\sum_{i=1}^{6} 2^{-\ell_i} = 2 \cdot 2^{-2} + 2 \cdot 2^{-3} + 2 \cdot 2^{-4} = \frac{7}{8} \leq 1 \ ,$$

Kraft's theorem implies that there is a prefix-free code with codewords of length $\ell_1, \ldots, \ell_6$. An example of such code is $C = \{00, 01, 100, 101, 1100, 1111\}$.

The conclusion of this section is contained in the following corollary:

Corollary 1.22. *Let $\ell_1, \ldots, \ell_m$ be a collection of positive integers. Then there exists a prefix-free code C with codewords of length $\ell_1, \ldots, \ell_m$, if and only if, $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$.*

Proof. It follows from Theorem 1.18 and Theorem 1.20. □

Remark. Fix $r \geq 3$ and consider r-ary codes; that is, codes over the alphabet $\Sigma = \{0, 1, \ldots, r-1\}$. Corollary 1.22 also holds, replacing $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$ by

$$\sum_{i=1}^{m} r^{-\ell_i} \leq 1 \ .$$

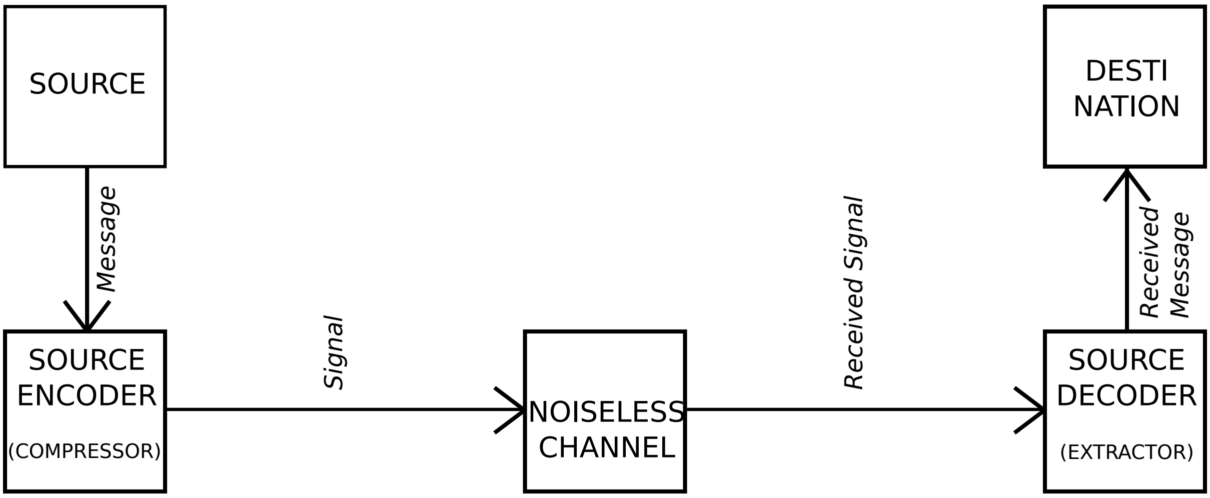
IMPORTANT CONCEPTS OF THIS CHAPTER

- Binary codes are subsets of bit-strings.
- Two important classes: prefix-free codes and uniquely decipherable codes.
- Every prefix-free code is uniquely decipherable, but the converse is not true.
- Prefix-free codes can be identified with edge-labelled binary trees.
- McMillan's theorem (Theorem 1.18) gives a necessary condition on the codewords lengths of a prefix-free code.
- Kraft's theorem (Theorem 1.20) gives a sufficient condition on a sequence of codewords lengths, such that there exists a prefix-free code with these codewords lengths.

Chapter 2

Noiseless channels

Noiseless communication scheme: In this chapter we consider the communication model where no errors occur during the transmission.



The *source alphabet* $S = \{s_1, \ldots, s_m\}$ is a finite set of symbols, called *letters*, used by source to write the message.

Some examples of source alphabets are:

- letters in the English alphabet, $S = \{a, b, c, d \ldots, z\}$.
- words in the English dictionary, $S = \{\text{aardvark}, \text{aardwolf}, \text{aaron}, \ldots, \text{zyzzyva}\}$.
- 24-bit pixel colors in RGB encoding, $S = \{(r, g, b) : r, g, b \in \mathbb{Z}, 0 \leq r, g, b < 256\}$.
- DNA sequence, $S = \{A, C, G, T\}$.

It is always more efficient (in terms of data transmission) to encode sources of large size; for instance encoding words instead of encoding letters. However, in this case the decoding scheme becomes too expensive to be used in practice. In real-world applications, a compromise between the two is used. For instance, it might be efficient to consider TH as an element of the source alphabet (T and H appear often together in English words) while considering AA as a source element barely increases the efficiency of the encoding.

2.1 Random Sources and Entropy

Let $\mathbf{p} = (p_1, \dots, p_m)$ be a discrete probability distribution. Recall that it must satisfy $p_i \geq 0$ for every $i \in [m]$ and $\sum_{i=1}^m p_i = 1$.

A random source $S = \{s_1, \dots, s_m\}$ has probability distribution $\mathbf{p} = (p_1, \dots, p_m)$ if the following is satisfied:

- the random source produces the letters one at a time,
- the random source produces the letter $s_i \in S$ with probability p_i, and
- the letter produced is independent of the past (*memoryless* source).

We associate a parameter, the so-called entropy, to a random source S that measures the "complexity" of the source.

Definition 2.1. (Entropy) Given a probability distribution $\mathbf{p} = (p_1, \dots, p_m)$, the *(binary) entropy* is defined as

$$H(\mathbf{p}) := -\sum_{i=1}^{m} p_i \log_2 p_i \ .$$

Given a random source S with probability distribution $\mathbf{p}$ we define its entropy as

$$H(S) := H(\mathbf{p}) \ .$$

Remark. It is possible to define the entropy replacing $\log_2$ by $\log_b$, where $b > 1$. Here it is convenient to use $\log_2$ since then the (binary) entropy of S is related to the number of bits we need to encode the letters of S. From now on, we will denote by log the binary logarithm $\log_2$.

Remark. Observe that the definition of entropy is equivalent to $H(S) = \sum_{i=1}^m p_i \log(1/p_i)$. We will use both expressions indistinctly. If one of the probabilities is 0, we let $0 \cdot \log 0 := \lim_{p \to 0} p \log p = 0$.

Example 2.2. If $S = \{s_1, s_2\}$ is a random source with $p_1 = p$ and $p_2 = 1 - p$, for some $p \in [0, 1]$, then

$$H(S) = H(p) := -p \log p - (1 - p) \log(1 - p) \ .$$

The function $H(p)$ is called the *binary entropy function.* Since we assumed that $0 \cdot \log 0 = 0$, the domain of $H(p)$ is $[0, 1]$. We have $H(0) = H(1) = 0$ and the maximum of $H(p)$ is attained when $p = 1/2$ (Exercise!).

Example 2.3. If $S_1 = \{s_1, \dots, s_m\}$ is a random source with $p_1 = 1$ and $p_2 = \dots = p_m = 0$ (i.e. the "random" source always produces s_1), then

$$H(S_1) = 1 \cdot (-\log 1) - (m - 1) \cdot 0 = 0 \ .$$

Observe that in this case the "random" source is deterministic and thus predictable (low entropy).

If $S_2 = \{s_1, \dots, s_m\}$ is a random source with $p_1 = p_2 = \dots = p_m = 1/m$ (i.e. the random source produces a uniform element of S_2), then

$$H(S_2) = -m \cdot \frac{1}{m} \cdot \log\left(\frac{1}{m}\right) = \log m \ .$$

In this case, the random source is highly unpredictable since every possible outcome appears with the same probability (high entropy).

The following technical statement will be useful later in the chapter.

Lemma 2.4 (Gibbs' Lemma). *Let $r_1, \ldots, r_m$ be a sequence of positive real numbers that satisfies $\sum_{i=1}^m r_i \leq 1$. Let $\mathbf{p} = (p_1, \ldots, p_m)$ be a probability distribution. Then*

$$H(\mathbf{p}) \leq \sum_{i=1}^m p_i \log\left(\frac{1}{r_i}\right).$$

Proof. Recall that $\ln x = (\ln 2) \log x$. If we multiply the inequality above by $\ln 2$, we obtain the following inequality

$$\sum_{i=1}^m p_i \ln\left(\frac{1}{p_i}\right) \leq \sum_{i=1}^m p_i \ln\left(\frac{1}{r_i}\right) .$$

Thus, it suffices to prove that $\sum_{i=1}^m p_i \left(\ln\left(\frac{1}{p_i}\right) - \ln\left(\frac{1}{r_i}\right)\right) \leq 0$. We have:

$$\begin{aligned}\sum_{i=1}^m p_i \left(\ln\left(\frac{1}{p_i}\right) - \ln\left(\frac{1}{r_i}\right)\right) &= \sum_{i=1}^m p_i \left(\ln\left(\frac{1}{p_i}\right) + \ln(r_i)\right)\\ &= \sum_{i=1}^m p_i \ln\left(\frac{r_i}{p_i}\right)\\ &\leq \sum_{i=1}^m p_i \left(\frac{r_i}{p_i} - 1\right).\end{aligned}$$

where in the last line we have used the inequality $\ln x \leq x - 1$, which holds for all $x > 0$ (the line $y = x - 1$ is the tangent of the curve $y = \ln x$ at $x = 1$ and $y = \ln x$ is a concave function). The latter is

$$\sum_{i=1}^m p_i \left(\frac{r_i}{p_i} - 1\right) = \sum_{i=1}^m r_i - \sum_{i=1}^m p_i = \sum_{i=1}^m r_i - 1 \leq 0,$$

by the assumption that $\sum_{i=1}^m r_i \leq 1$.

□

The next result shows that, in fact, the random source S_1 in Example 2.3 minimises the entropy among all random sources, while S_2 maximises it.

Theorem 2.5. *Let $\mathbf{p} = (p_1, \ldots, p_m)$ be a probability distribution. Then*

$$0 \leq H(\mathbf{p}) \leq \log m .$$

Proof. The lower bound follows since the entropy function is the sum of non-negative functions (recall that $\log p \leq 0$ for every $p \in (0, 1]$).

Let $r_i = 1/m$. So certainly $r_i > 0$ and $\sum_{i=1}^m r_i \leq 1$. Applying Gibbs' Lemma (Lemma 2.4) we obtain:

$$H(\mathbf{p}) \leq \sum_{i=1}^m p_i \log\left(\frac{1}{r_i}\right) = \sum_{i=1}^m p_i \log m = \log m \cdot \sum_{i=1}^m p_i = \log m .$$

□

2.2 Source encodings and their expected length

Definition 2.6 (Encoding, decoding)**.** A *source encoding scheme* (or simply, *encoding*) of S is an injective map $f : S \to \{0,1\}^*$, i.e. a map that assigns to each letter of S a unique string in $\{0,1\}^*$. The *code associated to the encoding* is the image of f in $\{0,1\}^*$ and is denoted by $C = C(f)$. We say that the encoding is *prefix-free* if the associated code is also prefix-free. For every letter $s \in S$, $f(s)$ is the *codeword of* s.

A *source decoding scheme* (or simply, *decoding*) is a map $g : \{0,1\}^* \to S$ such that $s = g(f(s))$ for every $s \in S$; i.e., the restriction of g to $C(f) \subseteq \{0,1\}^*$ is the inverse of f.

We can define a notion of efficiency for an encoding.

Definition 2.7. (Expected length of an encoding) Given a random source $S = \{s_1, \dots, s_m\}$ with probability distribution $\mathbf{p} = (p_1, \dots, p_m)$ and an encoding $f : S \to \{0,1\}^*$, consider the random variable $\ell(f)$ to be the length of the codeword $f(s)$, where s is a letter produced by the random source according to the probability distribution $\mathbf{p}$. Then, the *expected length of* f, denoted by $\mathbb{E}(\ell(f))$, is defined as the expected value of $\ell(f)$:

$$\mathbb{E}(\ell(f)) := \sum_{i=1}^{m} p_i |f(s_i)|.$$

Example 2.8. Let $S = \{s_1, s_2, s_3, s_4\}$ with $p_1 = 1/2$, $p_2 = 1/4$ and $p_3 = p_4 = 1/8$. The encoding $f_1(s_1) = 0, f_1(s_2) = 1, f_1(s_3) = 00$ and $f_1(s_4) = 01$, has expected length

$$\mathbb{E}(\ell(f_1)) = \frac{1}{2}\cdot|f_1(s_1)| + \frac{1}{4}\cdot|f_1(s_2)| + \frac{1}{8}\cdot|f_1(s_3)| + \frac{1}{8}\cdot|f_1(s_4)| = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 1 + \frac{1}{8}\cdot 2 + \frac{1}{8}\cdot 2 = \frac{5}{4}.$$

The encoding $f_2(s_1) = 1, f_2(s_2) = 00, f_2(s_3) = 010$ and $f_2(s_4) = 011$, has expected length

$$\mathbb{E}(\ell(f_2)) = \frac{1}{2}\cdot|f_2(s_1)| + \frac{1}{4}\cdot|f_2(s_2)| + \frac{1}{8}\cdot|f_2(s_3)| + \frac{1}{8}\cdot|f_2(s_4)| = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 = \frac{7}{4}.$$

We conclude that f_1 is more efficient than f_2.

As in Example 2.8, it is easy to design an encoding that minimises the expected length of a codeword: assign to the most probable letters the shortest strings of bits. However, the encoding obtained by this procedure is not likely to be prefix-free or uniquely decipherable. The goal of this chapter is to study prefix-free encodings of a random source with minimum expected length.

Definition 2.9 (Optimal prefix-free encoding)**.** Given a random source S, a prefix-free encoding f is *optimal* if it minimises $\mathbb{E}(\ell(f))$ among all the prefix-free encodings of S.

2.3 Shannon's Noiseless Encoding Theorem

The main message of this section is that the entropy function $H(S)$ approximates well the expected length of an optimal prefix-free encoding of S. This is known as *Shannon's Noiseless Encoding Theorem.*

Theorem 2.10 (Shannon, 1948)**.** *Let S be a random source and let $f : S \to \{0,1\}^*$ be an optimal prefix-free encoding of S. Then*

$$H(S) \le \mathbb{E}(\ell(f)) < H(S) + 1.$$

Remark. A very common mistake is the following: given a prefix-free encoding f, the condition $H(S) \leq \mathbb{E}(\ell(f)) < H(S)+1$ is necessary but not sufficient for f to be optimal.

Proof of Theorem 2.10. Let $S = \{s_1, \ldots, s_m\}$ be a random source with probability distribution $\mathbf{p} = (p_1, \ldots, p_m)$. We may assume all the $p_i > 0$. Let f be an optimal prefix-free encoding of S.

For every $i \in [m]$, let $\ell_i = |f(s_i)|$ be the length of the codeword of s_i and set $r_i = 2^{-\ell_i}$. Since $C(f)$ is prefix-free, by McMillan's Theorem (Theorem 1.18) we have

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

Thus, we can apply Gibbs' lemma (Lemma 2.4) to obtain the first inequality

$$H(S) = \sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) \leq \sum_{i=1}^{m} p_i \log\left(2^{\ell_i}\right) = \sum_{i=1}^{m} p_i \ell_i = \sum_{i=1}^{m} p_i |f(s_i)| = \mathbb{E}(\ell(f)) \ .$$

In order to prove the second inequality, we will construct a prefix-free encoding $\hat{f}$ that satisfies $\mathbb{E}(\ell(\hat{f})) < H(S) + 1$. Since f is an optimal prefix-free encoding, it minimises $\mathbb{E}(\ell(f))$, and the second inequality follows:

$$\mathbb{E}(\ell(f)) \leq \mathbb{E}(\ell(\hat{f})) < H(S) + 1 \ .$$

For every $i \in [m]$, let ℓ_i be the smallest integer such that $p_i \geq 2^{-\ell_i}$. Thus,

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

By Kraft's theorem (Theorem 1.20) there exists a prefix-free code $C = \{c_1, \ldots, c_m\}$ such that $|c_i| = \ell_i$. Consider the encoding $\hat{f}$ such that $\hat{f}(s_i) = c_i$ for every $i \in [m]$. Then $|\hat{f}(s_i)| = \ell_i$.

The definition of ℓ_i implies that $p_i < 2^{-(\ell_i - 1)}$, or equivalently, $\log\left(\frac{1}{p_i}\right) > \ell_i - 1$. We deduce that

$$H(S) = \sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) > \sum_{i=1}^{m} p_i(\ell_i - 1) = \sum_{i=1}^{m} p_i \ell_i - \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} p_i |\hat{f}(s_i)| - 1 = \mathbb{E}(\ell(\hat{f})) - 1 \ .$$

So

$$\mathbb{E}(\ell(\hat{f})) < H(S) + 1 \ ,$$

as desired. □

Remark. Shannon's proof is existential: it ensures the existence of an optimal encoding satisfying certain properties but it does not give a procedure to construct such encoding. This will be done in the next section using Huffman's encoding.

Sometimes, Shannon's theorem provides a way to identify optimal encodings, as shown in the following example.

Example 2.11. Let $S = \{s_1, s_2, s_3, s_4, s_5\}$ with probabilities $p_1 = p_2 = p_3 = 1/4$ and $p_4 = p_5 = 1/8$. Consider the prefix-free encoding

	s_1	s_2	s_3	s_4	s_5
$f(s_i)$	10	11	00	010	011

We have

$$\mathbb{E}(\ell(f)) = 3 \cdot \frac{1}{4} \cdot 2 + 2 \cdot \frac{1}{8} \cdot 3 = \frac{9}{4}$$
$$H(S) = 3 \cdot \frac{1}{4} \cdot \log_2(1/4) + 2 \cdot \frac{1}{8} \cdot \log_2(1/8) = \frac{9}{4} \ ,$$

and by Shannon's Noiseless Theorem, f is an optimal prefix-free encoding.

Remark. The lower bound in Theorem 2.10 is also valid for encodings f that produce a uniquely decipherable code $C(f)$. As we noted in the previous chapter, McMillan's theorem also holds for uniquely decipherable codes; that is, any uniquely decipherable encoding also satisfies $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$, and this is the only property of prefix-free codes that we use to prove that $H(S) \leq \mathbb{E}(\ell(f))$.

2.4 Huffman encoding

Shannon's Noiseless Encoding theorem shows the existence of a prefix-free encoding of a random source S with expected length between $H(S)$ and $H(S) + 1$. The following algorithm produces an optimal prefix-free encoding f_{H}, known as the *Huffman encoding*:

---

procedure HUFFMAN ENCODING
  Let $L = \{v_1, \ldots, v_m\}$ be a set of vertices where v_i corresponds to $s_i \in S$.
  Add the vertices of L into a tree T (the initial tree is a set of isolated vertices).
  while $|L| \geq 2$ **do**
    Pick two vertices v_{i_1} and v_{i_2} from L with the lowest probabilities.
    Create a new vertex v_{i_1,i_2} and assign probability $p_{i_1} + p_{i_2}$ to it.
    Add the edge $v_{i_1,i_2} \to v_{i_1}$ (with label 0) and $v_{i_1,i_2} \to v_{i_2}$ (with label 1) in T.
    Delete v_{i_1}, v_{i_2} from L and add v_{i_1,i_2}.
  end while
  Now $L = \{v\}$. We set $r = v$ to be the root of the tree T.
  For every $i \in [m]$, construct $f_{\mathrm{H}}(s_i)$ by concatenating the labels on the edges of the r-to-v_i path.
end procedure

Since $C(f_H)$ is the code associated to the edge-labelled binary tree T, by Lemma 1.14, it is a prefix-free code of size m.

Remark. Huffman discovered this encoding in 1952 when he was a graduate student at MIT. He was taking a course on Information Theory taught by Fano. Fano let the students choose between taking an exam, or reading and working on a paper about the problem of efficient encoding. Fano never told his students that the problem was unsolved. Huffman worked on it and almost gave up and started preparing for the exam. However, at the end he discovered the previous algorithm and proved that it gives an optimal encoding (Theorem 2.16). This improved the Shannon-Fano code which was the most efficient encoding known up to that date (see Exercise 1 in the Problem Sheet 2). Later Huffman said that if he would have known that his professor (Fano) did not managed to solve the problem, he would have given up. For further information, you can read: Gary Stix. Profile: David A. Huffman. *Scientific American*, 265(3):54, 58, September 1991.

Example 2.12. Let $S = \{s_1, s_2, \ldots, s_8\}$ be a random source with probability distribution

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
p_i	0.26	0.22	0.15	0.14	0.10	0.07	0.04	0.02

Step 1*:* Select s_7 and s_8, create their parent $s_{7,8}$ and assign probability $0.04 + 0.02 = 0.06$ to it.

	s_1	s_2	s_3	s_4	s_5	s_6	$s_{7,8}$
p_i	0.26	0.22	0.15	0.14	0.10	0.07	0.06

Step 2*:* Select s_6 and $s_{7,8}$, create their parent $s_{6,7,8}$ and assign probability $0.07+0.06 = 0.13$ to it.

	s_1	s_2	s_3	s_4	s_5	$s_{6,7,8}$
p_i	0.26	0.22	0.15	0.14	0.10	0.13

Step 3*:* Select $s_{6,7,8}$ and s_5, create their parent $s_{8,6,7,5}$ and assign probability $0.13 + 0.1 = 0.23$ to it.

	s_1	s_2	s_3	s_4	$s_{6,7,8,5}$
p_i	0.26	0.22	0.15	0.14	0.23

Step 4*:* Select s_3 and s_4, create their parent $s_{3,4}$ and assign probability $0.15 + 0.14 = 0.29$ to it.

	s_1	s_2	$s_{3,4}$	$s_{6,7,8,5}$
p_i	0.26	0.22	0.29	0.23

Step 5*:* Select $s_{6,7,8,5}$ and s_2, create their parent $s_{6,7,8,5,2}$ and assign probability $0.23 + 0.22 = 0.45$ to it.

	s_1	$s_{6,7,8,5,2}$	$s_{3,4}$
p_i	0.26	0.45	0.29

Step 5*:* Select $s_{3,4}$ and s_1, create their parent $s_{3,4,1}$ and assign probability $0.29+0.26 = 0.55$ to it.

	$s_{3,4,1}$	$s_{6,7,8,5,2}$
p_i	0.55	0.45

Step 6*:* Select $s_{3,4,1}$ and $s_{6,7,8,5,2}$, create their parent r, which will be the root of the tree.
A Huffman encoding of S is given by

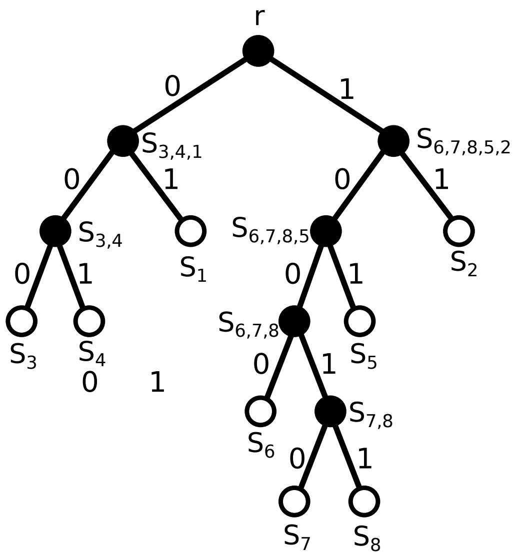
$f_{\mathrm{H}}(s_1)$	$f_{\mathrm{H}}(s_2)$	$f_{\mathrm{H}}(s_3)$	$f_{\mathrm{H}}(s_4)$	$f_{\mathrm{H}}(s_5)$	$f_{\mathrm{H}}(s_6)$	$f_{\mathrm{H}}(s_7)$	$f_{\mathrm{H}}(s_8)$
01	11	000	001	101	1000	10010	10011

The entropy of S is

$$\begin{aligned} H(S) = &-0.26 \log(0.26) - 0.22 \log(0.22) - 0.15 \log(0.15) - 0.14 \log(0.14) - 0.1 \log(0.1) \\ &- 0.07 \log(0.07) - 0.04 \log(0.04) - 0.02 \log(0.02) \\ \approx{}& 2.692\ , \end{aligned}$$

and the expected length of f_{H} is

$$\mathbb{E}(\ell(f_{\mathrm{H}})) = (0.26 + 0.22) \cdot 2 + (0.15 + 0.14 + 0.1) \cdot 3 + 0.07 \cdot 4 + (0.04 + 0.02) \cdot 5 = 2.71\ .$$



Remark.

1) The Huffman encoding is never unique. By convention, when merging two vertices v_{i_1} and v_{i_2} with probabilities p_{i_1} and p_{i_2}, respectively, with $p_{i_1} > p_{i_2}$, we always assign 0 to the edge connecting to v_{i_1} and 1 to the edge connecting to v_{i_2} (see Example 2.12). A different criterion will lead to a different code, but the length of each codeword would be the same.

2) There are random sources that have Huffman encodings with different codeword lengths. Observe that in step 5 of the Huffman encoding procedure (page 18) there can be more than two vertices with the lowest probabilities. In this case, different choices of v_{i_1} and v_{i_2} may lead to encodings with different codeword lengths. However, any of such encodings will have the same expected length.

Example 2.13. Let $S = \{s_1, s_2, s_3, s_4\}$ with $p_1 = p_2 = 1/3$ and $p_3 = p_4 = 1/6$.

In the first step of the Huffman encoding, we merge s_3 and s_4 into s_{34}. At this point, we have a new source $S = \{s_1, s_2, s_{34}\}$ with probability distribution $p_1 = p_2 = p_{34} = 1/3$. Consider the following two options,

- merge s_1 and s_2, which will lead to the Huffman encoding

	s_1	s_2	s_3	s_4
f_H^1	00	01	10	11

  Clearly, the expected length is $\mathbb{E}(\ell(f_H^1)) = 2$.

- merge s_2 and s_{34}, which will lead to the Huffman encoding

	s_1	s_2	s_3	s_4
f_H^2	1	00	010	011

  The expected length is

$$\mathbb{E}(\ell(f_H^2)) = 1 \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} + 3 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} = 2\ .$$

Exercise 2.14. One could think of another tree-based procedure to obtain an efficient encoding. Split the probability space in two sets having probability as equal as possible. Assign 0 to the first set, 1 to the other, and proceed recursively. Show that this algorithm produces a suboptimal encoding for the random source in Example 2.12.

In Example 2.12 we have seen that the Huffmann encoding has expected length close to the entropy of the source. The main result of this section states that, in fact, the Huffman encoding provides an optimal prefix-free encoding for a given random source.

Before proving the result we need the following proposition.

Proposition 2.15. *Let $S = \{s_1, \dots, s_m\}$ be a random source with probability distribution $\mathbf{p} = (p_1, \dots, p_m)$ and suppose that $p_1 \geq \cdots \geq p_m$. Then there exists an optimal prefix-free encoding f_* of S with $\ell_i = |f_*(s_i)|$ satisfying:*

a) $\ell_1 \leq \cdots \leq \ell_m$,

b) $\ell_{m-1} = \ell_m$, and

c) $f_*(s_{m-1})$ and $f_*(s_m)$ only differ in their last bit.

Proof. Let f_1 be an optimal prefix-free encoding. We can assume that the encoding f_1 satisfies the following: if $1 \leq i < j \leq m$ and $p_i = p_j$, then $|f_1(s_i)| \leq |f_1(s_j)|$. Otherwise, we can obtain it by reassigning the codewords among the letters with the same probability. This new encoding is also optimal since this reassignment preserves the value of $\mathbb{E}(\ell(f_1))$.

We first prove *a*), i.e. longer codewords are associated to letters with smaller probabilities.

For the sake of contradiction, suppose that $|f_1(s_i)| > |f_1(s_{i+1})|$, for some $i \in [m-1]$. By our previous assumption, it follows that $p_i > p_{i+1}$. Let f_2 be the encoding obtained from f_1 by swapping the codewords of s_i and s_{i+1}; that is $f_2(s_i) = f_1(s_{i+1})$, $f_2(s_{i+1}) = f_1(s_i)$ and $f_2(s_k) = f_1(s_k)$ for every $k \notin \{i, i+1\}$. One has

$$\begin{aligned}
\mathbb{E}(\ell(f_2)) &= \sum_{k=1}^{m} p_k |f_2(s_k)| \\
&= \left(\sum_{k=1}^{m} p_k |f_1(s_k)| \right) - p_i|f_1(s_i)| + p_i|f_2(s_i)| - p_{i+1}|f_1(s_{i+1})| + p_{i+1}|f_2(s_{i+1})| \\
&= \left(\sum_{k=1}^{m} p_k |f_1(s_k)| \right) - (p_i - p_{i+1})(|f_1(s_i)| - |f_1(s_{i+1})|) \\
&< \mathbb{E}(\ell(f_1)) \ ,
\end{aligned}$$

obtaining a contradiction to the fact that f_1 is optimal. This proves *a*).

We now prove a claim on the optimal prefix-free encoding:

<u>Claim 1</u>: *The code $C(f_1)$ associated to the encoding f_1 has two codewords of maximum length which differ only in their last bit.*

Proof of Claim 1: We first show that there are at least two codewords of maximum length. Suppose, for the sake of a contradiction, that $C(f_1)$ has only one word of maximum length. Since, f_1 is prefix-free, we could truncate the last bit of that word and still have a prefix-free encoding with smaller expected length - a contradiction as f_1 is optimal.

Now, assume that f_1 has at least two words of maximum length but no two of them differ only at their last bit. Then we could choose any two of them and truncate their last bit and get another prefix-free code (the new codewords are prefixes of the previous ones and cannot themselves be codewords already used) which has smaller expected length - again a contradiction. This concludes the proof of Claim 1.

We are ready to prove *b*) and *c*). By Claim 1, $C(f_1)$ has at least two codewords, say s_i and s_j, such that $f_1(s_i)$ and $f_1(s_j)$ have maximum length and differ only in their last bit. By *a*), $f_1(s_{m-1})$ and $f_1(s_m)$ also have maximum length. Thus, *b*) is satisfied. By reassigning the codewords $f_1(s_i)$ and $f_1(s_j)$ to s_{m-1} and s_m respectively, we can ensure *c*) also holds. □

Theorem 2.16. *Let $S = \{s_1, \dots, s_m\}$ be a random source with probability distribution $\mathbf{p} = (p_1, \dots, p_m)$. The Huffman encoding f_H is an optimal prefix-free code of S; that is, for every prefix-free encoding f of S we have*

$$\mathbb{E}(\ell(f_H)) \leq \mathbb{E}(\ell(f)) \ .$$

Proof. For a particular random source $S = \{s_1, \dots, s_m\}$, we denote by f_H^S its Huffman encoding. We will argue that f_H^S is optimal by induction on m.

The *base case* is for $m = 2$. In that case $S = \{s_1, s_2\}$ and the Huffman encoding is $f_{\mathrm{H}}^S(s_1) = 0$ and $f_{\mathrm{H}}^S(s_2) = 1$. The expected length of this encoding is 1 and it is clearly optimal since any codeword has length at least 1.

Suppose now that the statement of the theorem holds for all random sources with at most $m-1$ letters (this is the *induction hypothesis*). We will show that it also holds for any random source $S = \{s_1, \dots, s_m\}$ with probability distribution $\mathbf{p} = (p_1, \dots, p_m)$. We can assume that $p_1 \geq \dots \geq p_m$. Observe that such a property can be obtained by relabelling the elements of S.

Consider the modified source $R = \{s_1, \dots, s_{m-2}, s_{m-1,m}\}$ where the letter s_i occurs with probability p_i, for $i \in [m-2]$ and $s_{m-1,m}$ occurs with probability $p_{m-1} + p_m$. One can imagine $s_{m-1,m}$ as a new letter that represents both letters s_{m-1} and s_m.

Let f_*^S be the optimal prefix-free encoding of the random source S provided by Proposition 2.15. Construct a new encoding f_*^R for the random source R from f_*^S by assigning to $s_{m-1,m}$ the codeword obtained by removing the last bit from $f_*^S(s_{m-1})$ (or from $f_*^S(s_m)$). Note that since $f_*^S(s_m)$ and $f_*^S(s_{m-1})$ differed only in the last bit, and since f_*^S is prefix-free, f_*^R is well-defined and also prefix-free.

By the procedure used to construct the Huffman encoding f_{H}^S, the codewords $f_{\mathrm{H}}^S(s_{m-1})$ and $f_{\mathrm{H}}^S(s_m)$ differ only in their last bit. If we remove this last bit and assign to $s_{m-1,m}$ the remaining codeword, then we obtain a Huffman encoding of R, denoted by f_{H}^R. Since R has $m-1$ letters, by the induction hypothesis, f_{H}^R is an optimal encoding, that is

$$\mathbb{E}(\ell(f_{\mathrm{H}}^R)) \leq \mathbb{E}(\ell(f_*^R)) \tag{2.1}$$

But how does $\mathbb{E}(\ell(f_{\mathrm{H}}^R))$ compare with $\mathbb{E}(\ell(f_{\mathrm{H}}^S))$? Note that

$$\mathbb{E}(\ell(f_{\mathrm{H}}^S)) = \left(\sum_{i=1}^{m-2} p_i |f_{\mathrm{H}}^S(s_i)| \right) + p_{m-1}|f_{\mathrm{H}}^S(s_{m-1})| + p_m|f_{\mathrm{H}}^S(s_m)|,$$

and that

$$\mathbb{E}(\ell(f_{\mathrm{H}}^{R})) = \left(\sum_{i=1}^{m-2} p_i |f_{\mathrm{H}}^{R}(s_i)|\right) + (p_{m-1} + p_m)|f_{\mathrm{H}}^{R}(s_{m-1,m})|.$$

But for $i \in [m-2]$ we have $|f_{\mathrm{H}}^{S}(s_i)| = |f_{\mathrm{H}}^{R}(s_i)|$ and $|f_{\mathrm{H}}^{S}(s_{m-1})| = |f_{\mathrm{H}}^{S}(s_m)| = |f_{\mathrm{H}}^{R}(s_{m-1,m})| + 1$. Therefore,

$$\mathbb{E}(\ell(f_{\mathrm{H}}^{S})) = \mathbb{E}(\ell(f_{\mathrm{H}}^{R})) + (p_{m-1} + p_m)\ . \tag{2.2}$$

Following the same steps we can show that

$$\mathbb{E}(\ell(f_{*}^{S})) = \mathbb{E}(\ell(f_{*}^{R})) + (p_{m-1} + p_m)\ . \tag{2.3}$$

Using (2.1), (2.2) and (2.3), we obtain

$$\mathbb{E}(\ell(f_{\mathrm{H}}^{S})) = \mathbb{E}(\ell(f_{\mathrm{H}}^{R})) + (p_m + p_{m-1}) \leq \mathbb{E}(\ell(f_{*}^{R})) + (p_m + p_{m-1}) = \mathbb{E}(\ell(f_{*}^{S}))\ .$$

and since f_{*}^{S} is optimal, it follows that f_{H}^{S} also is.

$\square$

IMPORTANT CONCEPTS OF THIS CHAPTER

- In noiseless channels one is interested in measuring the efficiency of transmitting information.

- The entropy $H(S)$ of a random source S is a measure of randomness of S.

- The efficiency of an encoding f of S is measured by the expected length of a letter, denoted by $\mathbb{E}(\ell(f))$.

- Shannon's Noiseless theorem (Theorem 2.10) shows that the entropy is a lower bound for the efficiency ($H(S) \leq \mathbb{E}(\ell(f))$), but also shows that optimal encodings are close to attaining it ($\mathbb{E}(\ell(f)) < H(S) + 1$).

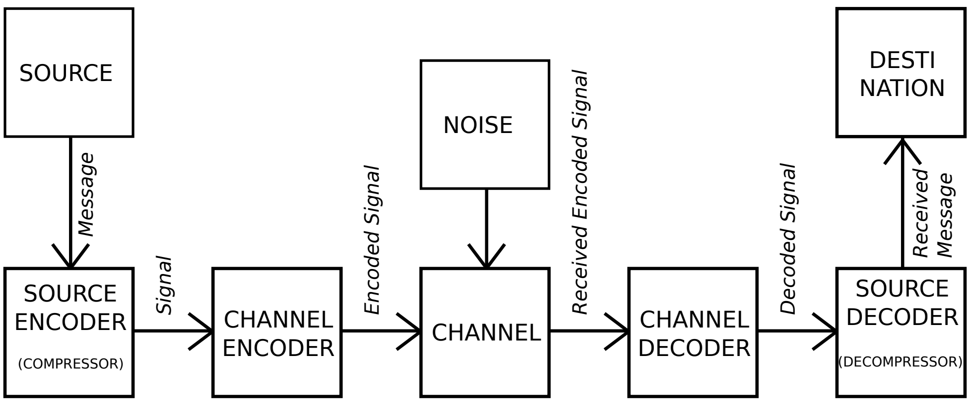
- Huffman's encoding f_H (see Section 2.4) gives a constructive way to obtain an optimal prefix-free encoding (Theorem 2.16).

Chapter 3

Noisy channels

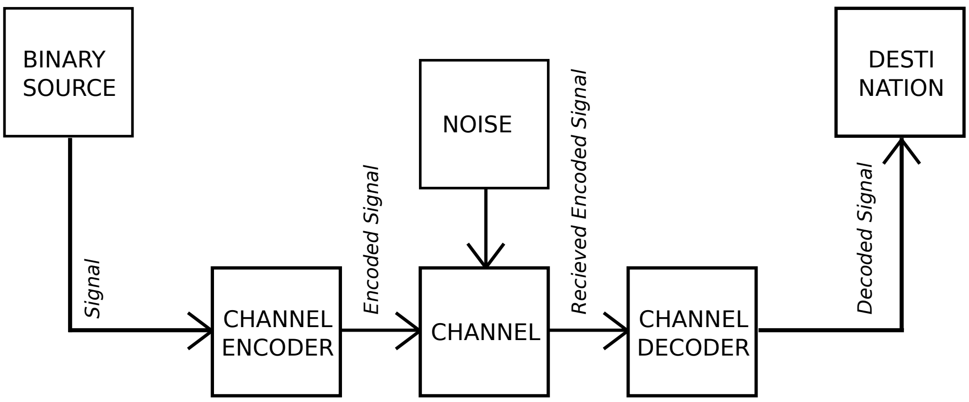
In this part of the course, we consider channels that produce *noise*. That is, occasionally the bits that are transmitted through the communication channel change, thus producing errors.

Full Noisy Communication Scheme:



Since the problem of efficient encoding was already studied in the previous section, from now on we will focus on the problem of robust encoding and we will forget about the source encoding. In real-life applications both efficiency and robustness are considered simultaneously.

Simplified Noisy Communication Scheme:



Definition 3.1 (noisy channel, symmetric, memoryless)**.** A *(binary) noisy channel* consists of a set of channel probabilities $\{p_{00}, p_{01}, p_{10}, p_{11}\}$, where p_{ij} is the probability that j is received if i has been sent. They must satisfy $p_{00} + p_{01} = p_{10} + p_{11} = 1$.

A noisy channel is *symmetric* if the probability of flipping a bit is p, for some $p \in [0, 1/2]$; that is, $p_{01} = p_{10} = p$ and $p_{00} = p_{11} = 1 - p$ and p is called the *crossover probability.*

A noisy channel is *memoryless* if the outcome of each transmission is independent of the outcome of the previous ones.

In what follows, we assume that the channel is binary, symmetric and memoryless.

Example 3.2. If $p = 0$ and 00000 is sent, the probability of receiving 00000 is $(1-p)^5 = 1$. If $p = 1/2$ and 00000 is sent, the probability of receiving any given string with i errors, $0 \leq i \leq 5$, is $p^i(1-p)^{5-i} = 2^{-5}$; that is, any string of length 5 is equally likely to be received. Thus, if $p = 0$ the channel is noiseless, while if $p = 1/2$ the channel is useless.

Example 3.3. For a binary string of length n, the number of errors that occur when it is transmitted through a binary symmetric channel with crossover probability p, follows a *binomial distribution* with n trials and probability p. In particular, the probability it contains exactly i errors, for $i \in [n]$, is $\binom{n}{i} p^i (1-p)^{n-i}$.

Definition 3.4 (Channel encoding, decoding, block code, transmission rate)**.** Given a finite set A and $n \in \mathbb{N}$, a *channel encoding scheme* (*encoding*) of A into strings of length n is an injective function $f : A \to \{0,1\}^n$. The *code* $C = C(f)$ is the image of f. In contrast to the previous part of the course, all codewords of C have length n. The *(channel) decoding scheme* (*decoding*) is a map $g : \{0,1\}^n \to A$ such that $g(f(x)) = x$ for every $x \in A$. However, we will mostly use the decoding as a map $h : \{0,1\}^n \to C$, such that $h(y) = y$ for every $y \in C$.

In applications, one usually considers $A = \{0,1\}^k$, for some $k \in [n]$. If so, C is called a *(block)* $[n,k]$*-code.*

For a code C of length n and a decoding h, the *probability of wrong decoding* is defined as

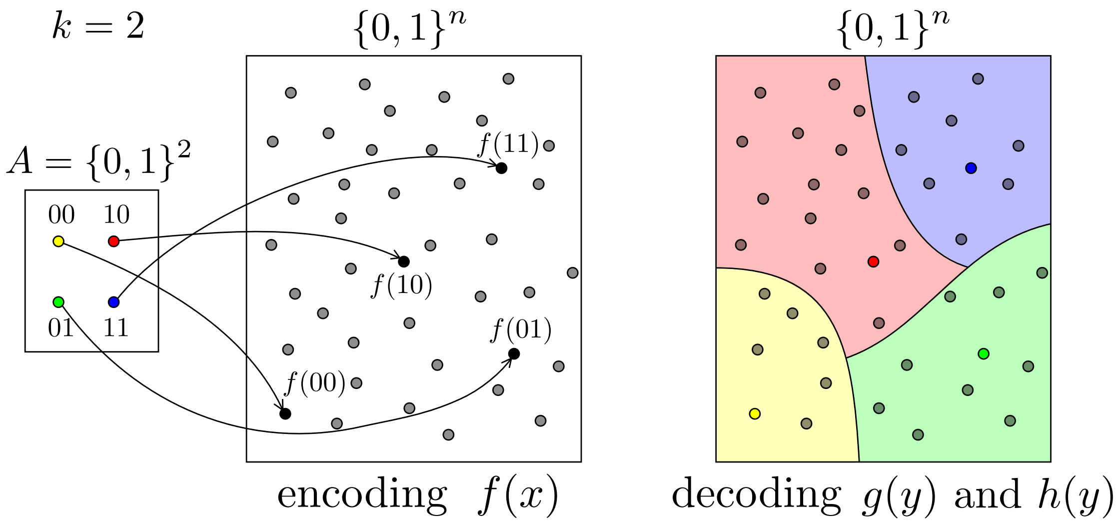
$$P_{\mathrm{err}}(C, h) := \max_{c \in C} \sum_{\substack{y \in \{0,1\}^n \\ h(y) \neq c}} \mathbb{P}(y \text{ is received} \mid c \text{ is sent})$$

The *transmission rate* of a code C of length n is the number of bits of information we would like to send, divided by the number of bits that are actually transmitted through the channel; that is,

$$R(C) = \frac{\log |C|}{n} .$$

In particular, if C is an $[n,k]$-code, then $R(C) = k/n$.

The main goal of Coding Theory is to design encoding/decoding schemes with a small probability of wrong decoding and a large transmission rate.



3.1 A first example: the repetition code

Consider the $[n, k]$-code with $k = 1$ and n a positive odd integer, given by the encoding $f_n : A = \{0, 1\} \to \{0, 1\}^n$ defined as

$$f_n(0) = \underbrace{00\ldots0}_{n}$$
$$f_n(1) = \underbrace{11\ldots1}_{n}$$

The block code associated to f_n is $C_n = \{00\ldots0, 11\ldots1\}$.

The decodings $g_n : \{0, 1\}^n \to \{0, 1\}$ and $h_n : \{0, 1\}^n \to C_n$ are defined as

$$g_n(w) = \begin{cases} 0 & \text{if } w \text{ contains more 0s than 1s} \\ 1 & \text{otherwise} \end{cases}$$
$$h_n(w) = \begin{cases} 00\ldots0 & \text{if } w \text{ contains more 0s than 1s} \\ 11\ldots1 & \text{otherwise} \end{cases}$$

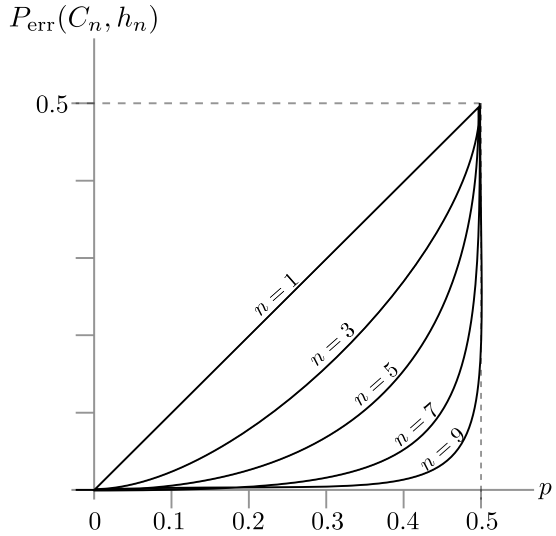
For example, if $n = 5$, then $g_5(01001) = 0$ and $h_5(01001) = 00000$. Note that $g_n(f_n(0)) = 0$ and $g_n(f_n(1)) = 1$, so g_n is a valid decoding for f_n, and similarly for h_n.

A string is wrongly decoded if and only if it contains more than $\frac{n+1}{2}$ errors. Thus, the probability of wrong decoding is:

$$P_{\text{err}}(C_n, h_n) = \mathbb{P}\left(\text{Bin}(n, p) \geq \frac{n+1}{2}\right) = \sum_{i=(n+1)/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} .$$

In fact, if $p < 1/2$, the probability of error decreases exponentially fast in n.

The transmission rate is $R(C_n) = 1/n$, for every bit of information that we would like to send, we transmit n bits.



n	$P_{err}(C_n, h_n)$	$P_{err}(C_n, h_n)$ for $p = 0.1$	$R(C_n)$
1	p	0.1	1
3	$p^3 + 3p^2(1-p)$	0.028	1/3
5	$p^5 + 5p^4(1-p)$ $+10p^3(1-p)^2$	0.0085	1/5
7	$p^7 + 7p^6(1-p)$ $+21p^5(1-p)^2$ $+35p^4(1-p)^3$	0.0027	1/7
9	$p^9 + 9p^8(1-p)$ $+36p^7(1-p)^2$ $+84p^6(1-p)^3$ $+126p^5(1-p)^4$	0.00089	1/9

For repetition codes, the choice of n is a trade-off. Large values of n give small probability of wrong decoding but make the transmission of each bit is slow (and expensive!), while for small values, the probability of an error increases but the transmission is faster.

In Chapter 4 we will cover Shannon's Noisy Encoding Theorem. This theorem shows the existence of codes that achieve arbitrarily small error probability and large transmission rate.

3.2 Basic properties of a code

In this section we introduce properties of codes as subsets of $\{0,1\}^n$. We introduce a metric in $\{0,1\}^n$ (Hamming dist.) and use it to define the minimum distance of a code.

Definition 3.5 (Hamming distance)**.** The *Hamming distance* between $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$, denoted by $d_H(x,y)$, is the number of places where the two strings differ:

$$d_H(x,y) := |\{i \in [n] : x_i \neq y_i\}| .$$

If $x = 01010$ and $y = 11000$, then $d_H(x,y) = 2$, since x and y only differ in the first and in the fourth position (see Figure 3.1).
The Hamming distance satisfies the axioms of a metric:

(1) *positive*: $d_H(x,y) \geq 0$ and $d_H(x,y) = 0$ if and only if $x = y$;

(2) *symmetric*: $d_H(x,y) = d_H(y,x)$;

(3) *triangle inequality*: $d_H(x,z) \leq d_H(x,y) + d_H(y,z)$.

Exercise 3.6. Prove that $d_H(x,y)$ satisfies (1), (2) and (3).

It has an interpretation in terms of Graph Theory.

Remark. The hypercube of dimension n, denoted by Q_n, is a graph with vertex set $V := \{0,1\}^n$ and edge set

$$E := \{xy : \; x, y \in \{0,1\}^n, \; d_H(x,y) = 1\} .$$

The metric space $(\{0,1\}^n, d_H)$ is isometric to (Q_n, d_G), where d_G is the graph distance.

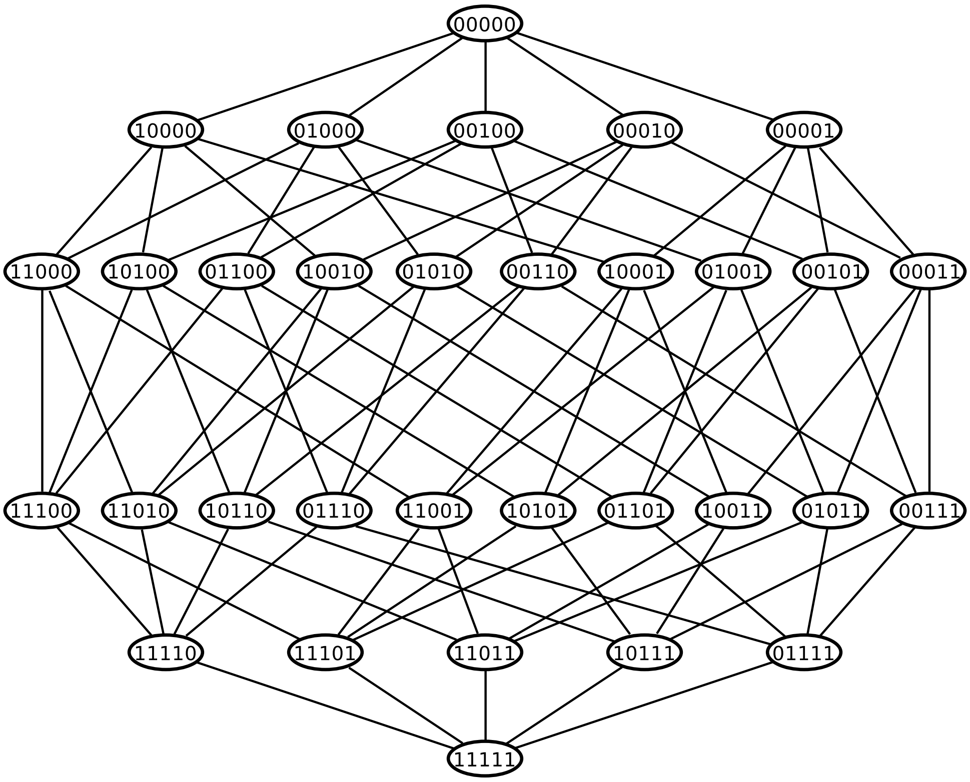


Figure 3.1: Hypercube Q_5

Definition 3.7 (Minimum distance)**.** The *minimum distance* of a code C, denoted by $d_H(C)$, is the minimum distance among all the pairs of codewords,

$$d_H(C) := \min_{\substack{c,c' \in C \\ c \neq c'}} d_H(c, c').$$

Example 3.8. The code $C = \{0000, 0101, 1010, 1111\}$ has minimum distance $d_H(C) = 2$.

Exercise 3.9. Compute $d_H(C)$ and $R(C)$ of the following codes:

i) $C = C_n$ is the repetition code of length $n \in \mathbb{N}$, n odd (Section 3.1).

ii) $C = \{x \in \{0,1\}^n : x = x_1 \dots x_n, x_1 + \cdots + x_n = 0 \bmod 2\}$

As suggested by these two exercises, codes with large minimum distance have a small transmission rate. However, these codes are more robust to errors, as we will see in the next section.

3.3 The minimum distance decoding

Equipped with the metric defined in the previous section, in this section we introduce the minimum distance decoding. This decoding provides an intuitive and general way to handle errors in a noisy environment.

Definition 3.10 (Minimum distance decoding)**.** A decoding $h : \{0,1\}^n \to C$ is a *minimum distance decoding* for C if for every $x \in \{0,1\}^n$ we have

$$d_H(x, h(x)) = \min_{c \in C} d_H(x, c)$$

In words, a minimum distance decoding assigns each string of length n to the closest codeword in C in terms of the Hamming distance. Sometimes, the codeword that minimises the distance is not unique; that is, there exist two codewords $c', c'' \in C$ with $d_H(x, c') = d_H(x, c'') = \min_{c \in C} d_H(x, c)$. In this case, the minimum distance decoding chooses one of them arbitrarily.

Example 3.11. Let $C = \{0000, 1110, 1101, 1011, 0111\}$. If h is a minimum distance decoding, then $h(1000) = 0000$ while $h(1100)$ is either 1110 or 1101.

Exercise 3.12. Let C be the code defined in Exercise 3.9. If h is a minimum distance decoding, compute all the possible values for $h(10110)$.

Definition 3.13. (Probability of wrong decoding using minimum distance decoding) Let C be a code of length n and let h be a minimum distance decoding for C. We define the *probability of wrong decoding of* C by

$$P_{\text{err}}(C) := P_{\text{err}}(C, h) = \max_{c \in C} \sum_{\substack{y \in \{0,1\}^n \\ h(y) \neq c}} p^{d_H(c,y)}(1-p)^{n-d_H(c,y)} .$$

Definition 3.14 (Error detecting/correcting code)**.** A code C is u *error-detecting*, if a minimum distance decoding can detect up to u errors. That is, if between 1 and u errors are made then one can identify that the string received is not the original codeword sent. A code C is v *error-correcting*, if a minimum distance decoding can correct up to v errors. That is, if at most v errors are made then one can still identify the original codeword sent.

Example 3.15. The code $C = \{000000, 111000, 000111, 111111\}$ is 2 error-detecting and 1 error-correcting.

The minimum distance of a code is related to these two parameters.

Proposition 3.16. *A code C is*

i) u error-detecting if and only if $d_H(C) \geq u + 1$.

ii) v error-correcting if and only if $d_H(C) \geq 2v + 1$.

Proof. We first prove i). Assume that C is u error-detecting. Since the code can detect any u errors, if there are two codewords $c_1, c_2 \in C$ such that $d_H(c_1, c_2) \leq u$, then if c_1 is transmitted but at most u errors are produced one could receive c_2 and no error would be detected. Conversely, if $d_H(C) \geq u + 1$, the codeword $c \in C$ is sent and at most u errors are produced, then the received word is not in C and we can identify it as an error.

Let us now prove ii). Assume first that $d_H(C) \geq 2v + 1$ and let us prove that C is v error-correcting. Suppose that one transmits a codeword $c_1 \in C$ and receives x, which satisfies $d_H(c_1, x) \leq v$, that is, at most v errors have occurred. Let $c_2 \in C$ be another codeword. By the assumption and the triangle inequality

$$2v + 1 \leq d_H(c_1, c_2) \leq d_H(c_1, x) + d_H(x, c_2) \leq v + d_H(x, c_2).$$

Thus, $d_H(x, c_2) \geq v+1 > v$. Therefore, a minimum distance decoding h satisfies $h(x) = c_1$ and C is v error-correcting.

Assume now that C is v error-correcting. For the sake of a contradiction, let us assume that $d_H(C) \leq 2v$. Then there exist two distinct codewords $c_1, c_2 \in C$ such that $d = d_H(c_1, c_2) \leq 2v$.

If d is even (that is, c_1 and c_2 differ in an even number of positions) then there exists $x \in \{0,1\}^n$ such that $d_H(c_1, x), d_H(c_2, x) = d/2 \leq v$. This word can be obtained from c_1 by flipping $d/2$ of the bits in the positions where c_1 and c_2 differ. If c_1 is transmitted but x is received due to noise, then at most v errors have occurred, but the minimum distance decoding is ambiguous; it can decode x to either c_1 or to c_2. So C is not v error-correcting.

If d is odd, in a similar way as before, there exists $x \in \{0,1\}^n$ such that $d_H(c_1, x) = d_H(c_2, x) + 1$. But as d is odd and $d \leq 2v$, it has to be the case that $d \leq 2v - 1$. So, $(d+1)/2 \leq v$. Then

$$d_H(c_1, x) = \frac{d+1}{2} \leq v \; .$$

If c_1 is transmitted but x is received, then at most v errors have occurred, but the minimum distance decoding rule will not decode x into c_1, as $d_H(c_2, x) < d_H(c_1, x)$ and $c_2 \in C$. So C is not v error-correcting.

In both cases, we reach a contradiction with the assumption that C was v error-correcting. □

3.4 Overview of Chapters 4 and 5

There are two ways to measure the robustness of codes.

	Chapter 4 (Shannon's approach)	Chapter 5 (Hamming's approach)
terminology	efficient codes	optimal codes
maximise	$R(C) = \frac{\log \lvert C \rvert}{n}$	$\lvert C \rvert$
constrained to	small $P_{\text{err}}(C)$	large $d_H(C)$
approach	Probabilistic	Combinatorial
results	Shannon's theorem	bounds on $A(n,d)$
handles	random noise	worst-case errors

IMPORTANT CONCEPTS OF THIS CHAPTER

- In noisy channels we are interested in transmitting the information in a robust way.
- There are two ways to measure robustness: by the probability of wrong decoding or by the number of bit-errors we can detect/correct.
- Using Hamming distance, we can see $\{0,1\}^n$ as a metric space; good codes are the ones where all elements are far from each other (minimum distance of the code).
- The minimum distance decoding provides a natural way to decode messages.
- The minimum distance of a code is directly related to the number of errors the code can correct/detect using a minimum distance decoding (Proposition 3.16).

Chapter 4

Efficient codes

In the previous chapter we have seen that the choice of n in the repetition code is a trade-off between efficiency and robustness. The goal of this chapter is to show that they are both attainable at the same time, that is, we show the existence of large codes that have arbitrarily small probability of wrong decoding.

We start with some technical bounds that will be useful later.

Definition 4.1 (Ball)**.** Let $0 \leq k \leq n$ and $x \in \{0,1\}^n$. The *ball of radius k centered at x* is the set of points $\{0,1\}^n$ at Hamming distance at most k from x, that is,

$$B_k^n(x) := \{y \in \{0,1\}^n \ : \ d_H(y,x) \leq k\} \ .$$

Write $b_k^n(x) := |B_k^n(x)|$.

The following result shows that $|B_k^n(x)|$ does not depend on x.

Proposition 4.2. *For any $0 \leq k \leq n$ and any $x \in \{0,1\}^n$ we have*

$$b_k^n := b_k^n(x) = \sum_{i=0}^{k} \binom{n}{i} \ .$$

Proof. The number of strings that differ from x in exactly i positions is $\binom{n}{i}$. □

Recall the *entropy function* defined in Example 2.2:

$$H(p) = -p \log p - (1-p) \log(1-p) \ .$$

One can use the entropy to bound the volume of a Hamming ball.

Proposition 4.3. *For every $\lambda \in [0, 1/2]$ and every $n \in \mathbb{N}$, we have*

$$\sum_{0 \leq i \leq \lambda n} \binom{n}{i} \leq 2^{H(\lambda) n}.$$

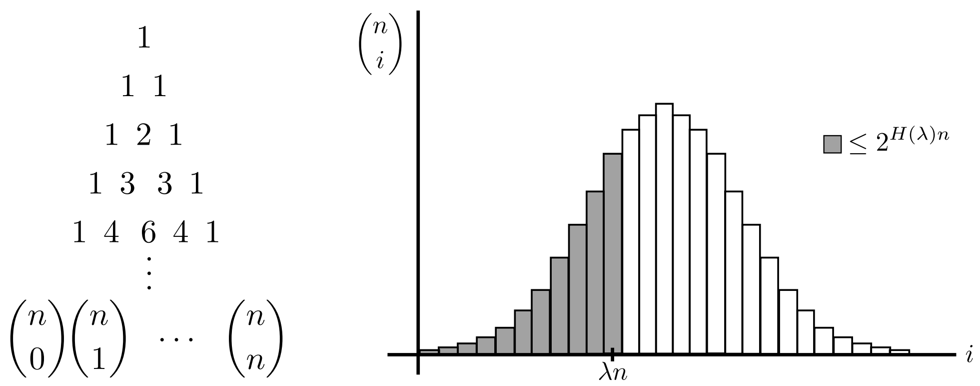
Proof. Recall the binomial theorem: $(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$. We write

$$\begin{aligned}
1 &= (\lambda + (1-\lambda))^n \\
&= \sum_{i=0}^{n} \binom{n}{i} \lambda^i (1-\lambda)^{n-i} \\
&\geq \sum_{0 \leq i \leq \lambda n} \binom{n}{i} \lambda^i (1-\lambda)^{n-i} \\
&= (1-\lambda)^n \sum_{0 \leq i \leq \lambda n} \binom{n}{i} \left(\frac{\lambda}{1-\lambda}\right)^i \\
&\geq (1-\lambda)^n \sum_{0 \leq i \leq \lambda n} \binom{n}{i} \left(\frac{\lambda}{1-\lambda}\right)^{\lambda n},
\end{aligned}$$

where in the last inequality we used that $\frac{\lambda}{1-\lambda} \in [0,1]$, since $\lambda \in [0, 1/2]$. Rearranging this, we obtain

$$\sum_{0 \leq i \leq \lambda n} \binom{n}{i} \leq \lambda^{-\lambda n} (1-\lambda)^{-(1-\lambda)n} = 2^{-(\lambda \log \lambda + (1-\lambda)\log(1-\lambda))n} = 2^{H(\lambda)n}.$$

□



4.1 The Sphere Covering Bound

The following result implies the existence of a code with certain minimum distance and large size.

Proposition 4.4 (The sphere covering bound)**.** *For any $1 \leq d \leq n$, there is a code C of length n with $d_H(C) \geq d$ and*

$$|C| \geq \frac{2^n}{b_{d-1}^n}.$$

Proof. We will construct the code C by greedily choosing codewords. Start with $C_0 := \emptyset$ and $V_0 := \{0,1\}^n$. Choose $c_1 \in V_0$ arbitrarily. Since the minimum distance between c_1 and any codeword of C different from c_1 in the final code should be at least d, any string

x with $d_H(x, c) \leq d-1$ cannot be in C. These words are precisely the ones in the ball $B^n_{d-1}(c_1)$. We set $V_1 := V_0 \setminus B^n_{d-1}(c_1)$ and $C_1 := \{c_1\}$. Independently of the choice of c_1, we have

$$|V_1| = 2^n - b^n_{d-1} \ .$$

Choose now $c_2 \in V_1$ arbitrarily. Note that $d_H(c_1, c_2) \geq d$ and thus $C_2 := \{c_1, c_2\}$ satisfies $d_H(C_2) \geq d$. If $V_2 := V_1 \setminus B^n_{d-1}(c_2)$, then

$$|V_2| \geq 2^n - 2b^n_{d-1} \ ,$$

where the inequality comes from the fact that there might be strings in $B^n_{d-1}(c_1) \cap B^n_{d-1}(c_2)$.

In general, assume that we have selected a code $C_{i-1} = \{c_1, \dots, c_{i-1}\}$ with $d_H(C_{i-1}) \geq d$ and that V_{i-1} is composed of the strings of length n at distance at least d from each of the elements in C_{i-1}. Choose $c_i \in V_{i-1}$ arbitrarily. Set $C_i := C_{i-1} \cup \{c_i\}$ and $V_i := V_{i-1} \setminus B^n_{d-1}(c_i)$. Then $d_H(C_i) \geq d$ and

$$|V_i| \geq 2^n - i \cdot b^n_{d-1} \ . \tag{4.1}$$

The process comes to its end when $V_i = \emptyset$. Let s be the smallest integer such that $V_s = \emptyset$. Then, by (4.1),

$$s \geq \frac{2^n - |V_s|}{b^n_{d-1}} = \frac{2^n}{b^n_{d-1}} \ .$$

The code $C := C_s$ has the correct size, and by construction, it satisfies $d_H(C) \geq d$.

□

We use the previous results in this section, to show the existence of efficient codes, where the trade-off between the minimum distance and the size is made explicit.

Theorem 4.5. *For every $\lambda \in [0, 1/2]$ and every $n \in \mathbb{N}$, there exists a code C of length n with $d_H(C) \geq \lambda n$ and*

$$|C| \geq 2^{(1-H(\lambda))n} \ ,$$

or, otherwise stated,

$$R(C) \geq 1 - H(\lambda) \ .$$

Proof. Let $d := \lceil \lambda n \rceil$. By Propositions 4.2 and 4.3, we have

$$b^n_{\lceil \lambda n \rceil - 1} = \sum_{0 \leq i \leq \lceil \lambda n \rceil - 1} \binom{n}{i} \leq \sum_{0 \leq i \leq \lambda n} \binom{n}{i} \leq 2^{H(\lambda) n} \ .$$

By Proposition 4.4 there exists a code C with minimum distance $d_H(C) \geq \lceil \lambda n \rceil \geq \lambda n$ and size

$$|C| \geq \frac{2^n}{b^n_{d-1}} = \frac{2^n}{b^n_{\lceil \lambda n \rceil - 1}} \geq 2^{(1-H(\lambda))n} \ ,$$

and

$$R(C) = \frac{\log |C|}{n} \geq 1 - H(\lambda) \ .$$

□

4.2 Shannon's Noisy Encoding Theorem

In the previous chapter we studied the error probability and the transmission rate of the repetition code. We saw that the choice of n is a trade-off between a robust code (small error probability) and a fast code (large transmission rate). In this section we will show the existence of codes with arbitrarily small error probability that achieve a high transmission rate of the channel. The only price we have to pay is to increase the length of the code.

Theorem 4.6 (Shannon, 1948). *For every $p \in [0, 1/2)$ and every $\epsilon \in (0, 1/2 - p]$, there exists an $n_0 = n_0(p, \epsilon)$ such that for all $n \geq n_0$, there exists a code C of length n with $P_{err}(C) < \epsilon$ and*

$$R(C) \geq 1 - H(p + \epsilon) .$$

Further topics. Is it possible to construct codes with larger transmission rate and arbitrarily small probability of wrong decoding? The answer is no: any code C with $R(C) > 1 - H(p)$ has $P_{\text{err}}(C)$ bounded away from 0. For this reason, $1 - H(p)$ is known as the capacity of the binary symmetric channel or the *Shannon limit*.

Further topics. A proof of Shannon's Theorem can be found in Section 2.2 of *Introduction to Coding Theory* from J.H. van Lint. The proof is non-constructive; it only shows the existence of a code satisfying the desired properties but does not provide an explicit way to construct it. A central open problem in coding theory is to construct codes that are as efficient as the ones provided by this theorem. Nowadays, one can reach transmission rates that are very close to Shannon's limit using turbocodes or LDPC codes. These codes are used in 4G mobile standards and in satellite communications.

Here we will prove a weaker version of Shannon's Theorem. This version has two advantages: it is constructive and n_0 is not as large as in Theorem 4.5; and one major drawback: it does not provide the best possible transmission rate.

Theorem 4.7. *For every $p \in [0, 1/4]$ and every $\epsilon \in (0, 1/2 - 2p]$, there exists an $n_0 = n_0(p, \epsilon)$ such that for all $n > n_0$, there exists a code C of length n with $P_{err}(C) < \epsilon$ and*

$$R(C) \geq 1 - H(2p + \epsilon) .$$

Proof. Let $n_0 := \lceil \frac{4p(1-p)}{\epsilon^3} \rceil$ and fix $n > n_0$. Let $\lambda := 2p + \epsilon$ and note that $\lambda \leq 1/2$. So by Theorem 4.5, there exists a code C of length n with $d_H(C) \geq \lambda n$ and $R(C) \geq 1 - H(\lambda)$. It suffices to show that $P_{\text{err}}(C)$ is small.

Let $d := \lceil \lambda n \rceil$. If we transmit a codeword $c_1 \in C$, but x is received and is decoded by a minimum distance decoding as $c_2 \in C$, this means that $d_H(x, c_2) \leq d_H(x, c_1)$. Using the triangle inequality we have

$$d \leq d_H(c_1, c_2) \leq d_H(x, c_1) + d_H(x, c_2) \leq 2d_H(x, c_1).$$

So

$$d_H(x, c_1) \geq d/2.$$

In other words, if a decoding error occurs, then $d_H(x, c_1) \geq d/2$. Let X be a binomially distributed random variable with parameters n and p. Given that c_1 is transmitted, x is a random word and $d_H(x, c_1)$ is the number of errors in the transmission, which is distributed as X.

Recall that $\lambda = 2p + \epsilon$, and thus $d/2 \geq \lambda n/2 \geq np + \epsilon n/2$. Thus

$$P_{\text{err}}(C) \leq \mathbb{P}(d_H(x, c_1) \geq d/2) = \mathbb{P}(X \geq d/2) \leq \mathbb{P}(X \geq np + \epsilon n/2) \; .$$

To bound the latter we use Chebyshev's inequality which states the following: let X be a random variable with expectation μ and variance σ^2, then for every $t > 0$

$$\mathbb{P}(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2} \; .$$

Since $\mu = \mathbb{E}(X) = np$ and $\sigma^2(X) = np(1-p)$, we set $t := \frac{\epsilon}{2\sqrt{p(1-p)}} \cdot \sqrt{n}$ to obtain

$$P_{\text{err}}(C) \leq \mathbb{P}(X \geq np + \epsilon n/2) \leq \mathbb{P}(|X - np| \geq \epsilon n/2) \leq \frac{1}{t^2} = \frac{4p(1-p)}{\epsilon^2 n} < \epsilon \; ,$$

where the last inequality follows from the choice of $n > n_0$. □

Note that, since Theorem 4.5 is constructive, so we can construct the code in Theorem 4.7.

IMPORTANT CONCEPTS OF THIS CHAPTER

- The sphere covering bound (Proposition 4.4) constructs codes that have large size and large minimum distance.
- Shannon's Noisy Theorem (Theorem 4.5) shows the existence of codes that are efficient for the binary symmetric channel; that is, codes with arbitrarily small probability of wrong decoding and maximum transmission rate. While its proof is existential, nowadays we have codes that are almost efficient.
- One can use the sphere covering bound, to obtain a weak constructive statement of Shannon's Noisy Theorem (Theorem 4.7).

Chapter 5

Bounds on codes

In the previous chapter we wanted to maximise $R(C)$ (or $|C|$) and minimise $P_{\mathrm{err}}(C)$ simultaneously. The probability of wrong decoding is closely related to the minimum distance of the code. Another approach at good codes is to fix the length n and the minimum distance d of a code, and ask for its maximum size.

Definition 5.1 ((n,d)- and (n,M,d)-codes)**.** For integers $1 \le d \le n$, a code C is an (n,d)-*code* if it has length n and minimum distance $d_H(C) \ge d$. An (n,M,d)-*code* is an (n,d)-code of size M.

Definition 5.2 ($A(n,d)$, Optimal codes)**.** For integers $1 \le d \le n$, let $A(n,d)$ be the largest M such that there exists an (n,M,d)-code. An (n,d)-code is *optimal* if has size $A(n,d)$.

A wide open question in Coding Theory is to determine $A(n,d)$ for all values of n and d. To show that $A(n,d) = M$, we need to prove two things:

(1) $A(n,d) \le M$: any (n,d)-code C satisfies $|C| \le M$,

(2) $A(n,d) \ge M$: there exists an (n,M,d)-code.

Here we give a simple example.

Example 5.3. We have $A(n,1) = 2^n$.

(1) $A(n,1) \le 2^n$: trivially, since there are at most 2^n codewords of length n.

(2) $A(n,1) \ge 2^n$: consider the code $C = \{0,1\}^n$ of length n, $d_H(C) \ge 1$ and $|C| = 2^n$.

5.1 Operations on Codes

There are several operations that can be performed to obtain new codes from other ones. These operations are extremely useful to derive bounds for $A(n,d)$.

In order to define some of these operations, we need to understand $\{0,1\}^n$ as a vector space of dimension n over the binary field $\mathbb{F}_2$, usually denoted by $\mathbb{F}_2^n$. This consists of all vectors of dimension n whose coordinates are in $\{0,1\}$. Addition of any two vectors is pointwise modulo 2. For instance, $(01001) + (01110) = (00111)$.

5.1.1 The sum code

Definition 5.4 (Sum code)**.** Given a code C and $x \in \mathbb{F}_2^n$, the *sum code* is defined as

$$C + x := \{c + x : c \in C\} .$$

Proposition 5.5. *For every $1 \leq d \leq n$, there exists an optimal (n, d)-code C with $(00 \dots 0) \in C$.*

Proof. Let C_0 be an optimal (n, d)-code, that is $|C_0| = A(n, d)$. Let $c_0 \in C_0$ and consider the code $C = C_0 + c_0$. Then C is also an optimal (n, d)-code, and $(00 \dots 0) = c_0 + c_0 \in C$. $\square$

Example 5.6. We have $A(4, 3) = 2$.

(1) $A(4, 3) \leq 2$: For the sake of contradiction, suppose there exists a code C of length 4, $d_H(C) \geq 3$ and $|C| \geq 3$. By Proposition 5.5, we can assume that $0000 \in C$. The only other possible elements of C are $0111, 1110, 1101, 1011$ and 1111, however each pair of them has Hamming distance at most 2, thus reaching a contradiction.

(2) $A(4, 3) \geq 2$: consider $C = \{0000, 1110\}$ of length 4, $d_H(C) \geq 3$ and $|C| = 2$.

5.1.2 The punctured code

Definition 5.7 (Punctured code)**.** Given a code C of length n and $1 \leq \ell \leq n$, the *punctured code* $C^*(\ell)$ of C is obtained by truncating the last ℓ bits of each codeword of C; that is,

$$C^*(\ell) := \{x_1 \dots x_{n-\ell} : c = x_1 \dots x_n, c \in C\} .$$

If C is an (n, M, d)-code and $\ell < d$ (so $d \geq 2$), then $C^*(\ell)$ is an $(n - \ell, M, d - \ell)$-code. This is useful to bound $A(n, d)$, as we show in the following example.

Example 5.8. We have $A(n, 2) = 2^{n-1}$.

(1) $A(n, 2) \leq 2^{n-1}$: Let C be an $(n, 2)$-code. Consider the punctured code $C^*(1)$ obtained by truncating the last position. Since $d_H(C) \geq 2$, we have that $d_H(C^*(1)) \geq 1$ and that $|C| = |C^*(1)|$. However, the punctured code has length $n - 1$ so it contains at most 2^{n-1} codewords. We conclude that $|C| = |C^*(1)| \leq 2^{n-1}$.

(2) $A(n, 2) \geq 2^{n-1}$: Consider the code

$$C = \{c \in \{0, 1\}^n : c = x_1 \dots x_n, x_1 + \dots + x_n = 0 \bmod 2\} .$$

Then, C has length n. We have $d_H(C) \geq 2$ since there are no two codewords in the code differing in only one position; otherwise one of them would satisfy $x_1 + \dots + x_n = 1 \bmod 2$. The code satisfies $|C| \geq 2^{n-1}$ since exactly half of the words in $\{0, 1\}^n$ have an even number of ones.

5.1.3 The parity check code

We first define the notion of weight.

Definition 5.9 (Weight). The *weight* of $x \in \{0,1\}^n$ is the number of 1s in x and is denoted by $w(x)$.

One can use the weights of x and y, to express their Hamming distance. If $a(x,y)$ is the number of positions where x and y are both 1, then

$$d_H(x,y) = w(x) + w(y) - 2a(x,y)\ . \tag{5.1}$$

Definition 5.10 (Parity check code). Given a code C, the *parity check code* $\overline{C}$ of C is obtained by appending a 0 at the end of $c \in C$ if the number of 1s in c is even; or a 1 if the number of 1s in c is odd. That is,

$$\overline{C} := \{x_1 \dots x_n x_{n+1} : \, c = x_1 \dots x_n,\, c \in C,\, x_{n+1} = (w(c) \bmod 2)\}\ .$$

We have the following result on the minimum distance of the parity check code.

Proposition 5.11. *Given a code C,*

i) $d_H(\overline{C})$ is even;

ii) if $d_H(C)$ is odd, then $d_H(\overline{C}) = d_H(C) + 1$.

Proof. To prove i) consider two different codewords $c_1, c_2 \in C$. Observe that $\bar{c}_1$ and $\bar{c}_2$ have even weight. By (5.1), we have

$$d_H(\bar{c}_1, \bar{c}_2) = w(\bar{c}_1) + w(\bar{c}_2) - 2a(\bar{c}_1, \bar{c}_2),$$

and since all the terms above are even, we conclude that $d_H(\bar{c}_1, \bar{c}_2)$ is even.

To prove ii) suppose now that $d_H(C)$ is odd. Clearly, for any $c_1, c_2 \in C$, we have $d_H(\bar{c}_1, \bar{c}_2) \le d_H(c_1, c_2) + 1$, so $d_H(\overline{C}) \le d_H(C) + 1$.

Let $c_1, c_2 \in C$. If $d_H(c_1, c_2) = d_H(C)$, by part one $d_H(\bar{c}_1, \bar{c}_2)$ is even, so $d_H(\bar{c}_1, \bar{c}_2) \ge d_H(c_1, c_2) + 1 = d_H(C) + 1$. If $d_H(c_1, c_2) > d_H(C)$, then $d_H(\bar{c}_1, \bar{c}_2) \ge d_H(c_1, c_2) \ge d_H(C) + 1$. It follows that $d_H(\overline{C}) \ge d_H(C) + 1$, concluding the proof. □

An interesting consequence of the previous proposition is the following.

Theorem 5.12. *If d is odd, then $A(n,d) = A(n+1, d+1)$.*

Proof. Let C be an optimal (n,d)-code. So $|C| = A(n,d)$. Since $d = d_H(C)$ is odd, by the second part of Proposition 5.11, the parity check code $\overline{C}$ is an $(n+1, d+1)$-code with $|\overline{C}| = A(n,d)$, which implies that $A(n+1, d+1) \ge A(n,d)$.

Suppose now that C is an optimal $(n+1, d+1)$-code. So $|C| = A(n+1, d+1)$. Consider the code $C^*(1)$ of length n obtained by truncating the last bit of C. Since $d+1 \ge 2$, $|C^*(1)| = |C| = A(n+1, d+1)$. Moreover, $d_H(C^*(1)) \ge d_H(C) - 1 = d$. Thus, $C^*(1)$ is an (n,d)-code with $|C^*(1)| = A(n+1, d+1)$, which implies that $A(n,d) \ge A(n+1, d+1)$. □

Example 5.13. We have $A(5,4) = 2$. Indeed, in Example 5.6 we have shown that $A(4,3) = 2$. Since 3 is odd, by Theorem 5.12, we have $A(5,4) = A(4,3) = 2$.

5.2 Upper bounds on $A(n, d)$

Determining $A(n, d)$ for all values of n and d is an open problem in Mathematics. Thus, it is interesting to find general bounds on them. In Proposition 4.4 we have shown the existence of a large code of certain length with large minimum distance, which translates to a lower bound of the form

$$A(n, d) \geq \frac{2^n}{b^n_{d-1}} .$$

In this section, we will focus on the upper bounds.

Theorem 5.14 (The sphere packing bound; or Hamming bound)**.** *For every $1 \leq d \leq n$, we have*

$$A(n, d) \leq \frac{2^n}{b^n_{\lfloor \frac{d-1}{2} \rfloor}}.$$

Proof. Let C be an (n, d)-code and let $t := \lfloor \frac{d-1}{2} \rfloor$. For any two codewords $c_1, c_2 \in C$, we have $d_H(c_1, c_2) \geq d$. Thus, $B^n_t(c_1) \cap B^n_t(c_2) = \emptyset$. Indeed, if not, then let $z \in B^n_t(c_1) \cap B^n_t(c_2)$ and by the triangle inequality, we would obtain a contradiction

$$d \leq d_H(c_1, c_2) \leq d_H(c_1, z) + d_H(z, c_2) \leq t + t = 2t \leq 2 \cdot \frac{d-1}{2} = d - 1 .$$

It follows that

$$2^n = |\{0,1\}|^n \geq \sum_{c \in C} |B^n_t(c)| = |C| \cdot b^n_t ,$$

from where we conclude

$$|C| \leq \frac{2^n}{b^n_{\lfloor \frac{d-1}{2} \rfloor}} .$$

□

Example 5.15. We have that $A(7,3) = 16$.

(1) $A(7,3) \leq 16$: we will use Hamming's bound. We have $\lfloor \frac{d-1}{2} \rfloor = 1$ and

$$b^7_1 = \binom{7}{0} + \binom{7}{1} = 8 .$$

Using Hamming's bound,

$$A(7,3) \leq \frac{2^7}{8} = 16 .$$

(2) $A(7,3) \geq 16$: the Hamming $[7,4]$-code constructed in Exercise 3 from Problem Sheet 3 is a $(7,16,3)$-code.

Hamming bound is quite meaningful for small values of d, but not very good for large ones. The next upper bound works well for large values of d.

Theorem 5.16 (Plotkin's bound)**.** *For every* $1 \le d \le n$ *with* $2d > n$*, we have*

$$A(n,d) \le \frac{2d}{2d-n} \ .$$

Proof. Let C be an (n,d)-code of size m and $C = \{c_1, \dots, c_m\}$. We will provide upper and lower bounds for the following quantity,

$$S := \sum_{1 \le i < j \le m} d_H(c_i, c_j) \ .$$

For any two codewords $c_i, c_j \in C$, we have $d_H(c_i, c_j) \ge d$. It follows that

$$S = \sum_{1 \le i < j \le m} d_H(c_i, c_j) \ge \sum_{1 \le i < j \le m} d = \binom{m}{2} d \ . \tag{5.2}$$

For the upper bound, we first write $d_H(c_i, c_j) = \sum_{k=1}^{n} \mathbf{1}(c_{ik} \ne c_{jk})$, where c_{ik} is the k-th bit of c_i and

$$\mathbf{1}(c_{ik} \ne c_{jk}) = \begin{cases} 1, & \text{if } c_{ik} \ne c_{jk}; \\ 0, & \text{if } c_{ik} = c_{jk} \end{cases} .$$

Hence, we can write

$$S = \sum_{1 \le i < j \le m} d_H(c_i, c_j) = \sum_{1 \le i < j \le m} \sum_{k=1}^{n} \mathbf{1}(c_{ik} \ne c_{jk}) = \sum_{k=1}^{n} \sum_{1 \le i < j \le m} \mathbf{1}(c_{ik} \ne c_{jk}) \ .$$

For a given $k \in [n]$, let ℓ_k be the number of 0s in the k-th bit of the codewords in C. Then,

$$\sum_{1 \le i < j \le m} \mathbf{1}(c_{ik} \ne c_{jk}) = \ell_k (m - \ell_k) \le \frac{m^2}{4} \ .$$

It follows that

$$S \le n \cdot \frac{m^2}{4} \ . \tag{5.3}$$

Thus, (5.2) and (5.3) yield

$$\binom{m}{2} d \le S \le n \cdot \frac{m^2}{4} \ .$$

We can write $\binom{m}{2} = \frac{m(m-1)}{2} = \frac{m^2}{2}\left(1 - \frac{1}{m}\right)$. So the above inequality becomes

$$\left(1 - \frac{1}{m}\right) d \le \frac{n}{2}.$$

From straightforward computations, we obtain that $m \le \frac{2d}{2d-n}$, as desired. □

Example 5.17. We have $A(3m, 2m) = 4$.

(1) $A(3m, 2m) \le 4$: note that $n = 3m$ and $d = 2m$. Since $2d = 4m > 3m = n$, we can apply Plotkin's inequality to obtain

$$A(3m, 2m) \le \frac{2d}{2d-n} = \frac{4m}{4m-3m} = 4 \ .$$

(2) $A(3m, 2m) \geq 4$: consider the code $C = \{c_1, c_2, c_3, c_4\}$ where

$$\begin{aligned} c_1 &= 00\ldots0\,|\,00\ldots0\,|\,00\ldots0 \\ c_2 &= 11\ldots1\,|\,11\ldots1\,|\,00\ldots0 \\ c_3 &= 11\ldots1\,|\,00\ldots0\,|\,11\ldots1 \\ c_4 &= 00\ldots0\,|\,11\ldots1\,|\,11\ldots1\,. \end{aligned}$$

This is a $(3m, 4, 2m)$-code.

IMPORTANT CONCEPTS OF THIS CHAPTER

- If our measure of robustness is the minimum distance of a code, our goal is to determine the largest size of a code of length n and minimum distance at least d, denoted by $A(n, d)$ (optimal (n, d)-code).
- Determining $A(n, d)$ is an extremely difficult problem which is wide open.
- To bound $A(n, d)$ from above, we need to provide a proof that applies to all codes with such parameters.
- To bound $A(n, d)$ from below, we need to construct an (n, d)-code of a given size.
- One can use operations on codes to transfer known results on $A(n, d)$ to other values of n or d (see Section 5.1).
- General bounds such as the Hamming bound (Theorem 5.14) and the Plotkin's bound (Theorem 5.16) help us to understand $A(n, d)$.

Chapter 6

Linear codes

As in some parts of the previous chapter, here we understand the codewords as elements of $\mathbb{F}_2^n$, the vector space of dimension n over the binary field $\mathbb{F}_2$.

Definition 6.1 (Linear Code)**.** A *linear* $[n, k]$*-code* is a subspace of $\mathbb{F}_2^n$ of dimension k. We call n the *length* and k the *dimension* or *rank* of the code.

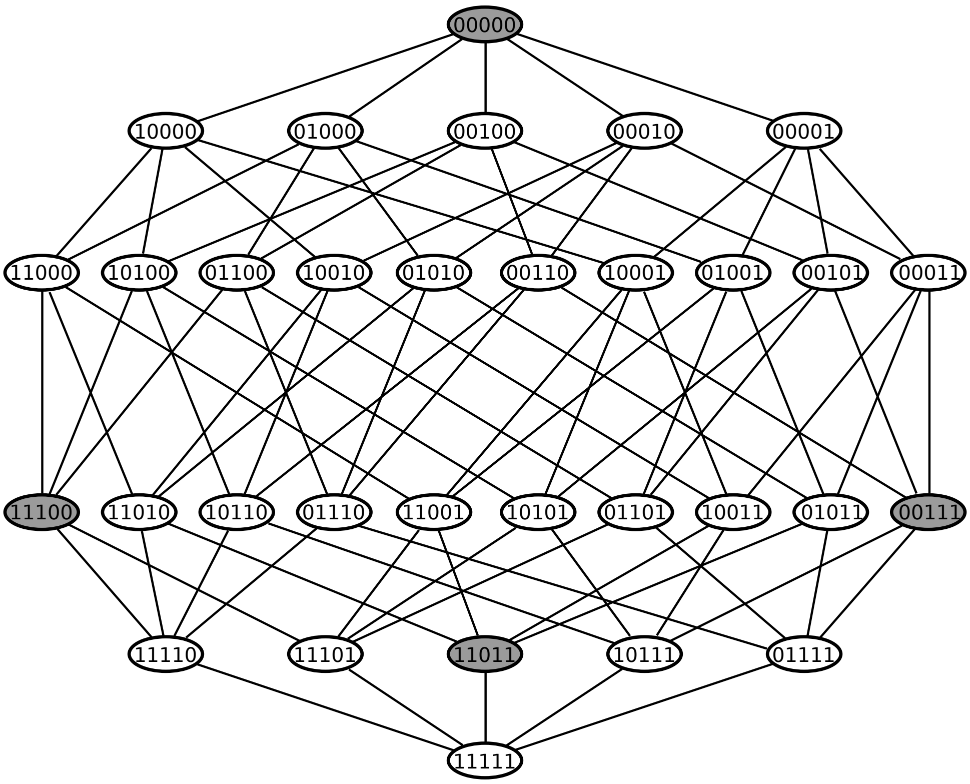


Figure 6.1: A linear $[5, 2]$-code generated by (11100) and (00111).

Remark. Recall that a subset C of $\mathbb{F}_2^n$ is a subspace if and only if

1) it is closed under addition, that is $c_1, c_2 \in C$ implies $c_1 + c_2 \in C$;

2) it is closed under multiplication by elements of $\mathbb{F}_2$, that is $c \in C$ and $\lambda \in \mathbb{F}_2$, implies $\lambda c \in C$;

3) C contains the zero vector.

A nice property of $\mathbb{F}_2^n$ is that, to check C is a subspace, we in fact only need to check 1). Indeed, suppose 1) holds. Then given any $c_1 \in C$ we have that $0 = c_1 + c_1 \in C$; so 3) holds. Further given $c \in C$, then if $\lambda = 1$, $\lambda c = c \in C$ and if $\lambda = 0$, $\lambda c = 0 \in C$; so 2) holds.
Recall that the dimension/rank of C is the size of a basis of C.

The binary pointwise addition of two vectors allows us to express the Hamming distance between two vectors in terms of the weight of their sum. Recall that the weight of x, denoted by $w(x)$, is the number of 1s in x.

Lemma 6.2. *For any $x, y \in \mathbb{F}_2^n$, we have $d_H(x, y) = w(x + y)$.*

Proof. The vector $x + y$ has 0 at the positions where x and y coincide and 1 at the positions where they differ. Hence, the number of ones in $x + y$ (its weight) is equal to the Hamming distance between x and y. □

We present two examples of linear codes.

Example 6.3. Let

$$C_1 := \{x \in \mathbb{F}_2^n : w(x) = 0 \bmod 2\} .$$

Let us check that C_1 is a subspace of $\mathbb{F}_2^n$. It suffices to prove that if $x, y \in C_1$, then $x + y \in C_1$. Using Lemma 6.2 and (5.1), we have

$$w(x + y) = d_H(x, y) = w(x) + w(y) - 2a(x, y) ,$$

where $a(x, y)$ is the number of places where both x and y have 1s. Since $x, y \in C_1$, $w(x)$ and $w(y)$ are even, so $w(x + y)$ is also even. We conclude that $x + y \in C_1$ and C_1 is a linear code.

Example 6.4. Let n be even and set

$$C_2 := \{x \in \mathbb{F}_2^n : x = x_1x_2 \dots x_n,\ x_1 = x_2,\ x_3 = x_4, \dots, x_{n-1} = x_n\} .$$

Let us check that C_2 is a subspace. It suffices to prove that if $x, y \in C_2$, then $x + y \in C_2$. Write $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$. We have $x_1 = x_2$, $x_3 = x_4, \dots, x_{n-1} = x_n$ and $y_1 = y_2$, $y_3 = y_4, \dots, y_{n-1} = y_n$, which implies that $x_1 + y_1 = x_2 + y_2$, $x_3 + y_3 = x_4 + y_4, \dots, x_{n-1} + y_{n-1} = x_n + y_n$. Thus, $x + y = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n) \in C_2$ and C_2 is a linear code.

Lemma 6.2 also allows us to express the minimum distance of the linear code C in terms of the minimum weight among all non-zero codewords.

Proposition 6.5. *If C is a linear code, then*

$$d_H(C) = \min_{\substack{c \in C \\ c \neq 0}} w(c) .$$

Proof. Let $x, y \in C$ such that $d_H(x, y) = d_H(C)$, and recall that $x \neq y$. By Lemma 6.2, we have $d_H(x, y) = w(x + y)$. But $x + y \in C$, and, since $x + y \neq 0$, it follows that

$$d_H(C) = d_H(x, y) = w(x + y) \geq \min_{\substack{c \in C \\ c \neq 0}} w(c) .$$

Let $\hat{c} \in C$ such that $w(\hat{c}) = \min_{\substack{c \in C \\ c \neq 0}} w(c)$. Since $0 \in C$ and by Lemma 6.2

$$\min_{\substack{c \in C \\ c \neq 0}} w(c) = w(\hat{c}) = w(\hat{c} + 0) = d_H(\hat{c}, 0) \geq d_H(C) \ .$$

□

Algorithmic Remark. Proposition 6.5 is interesting from an algorithmic point of view. Suppose that we want to compute $d_H(C)$ for a code C of size m, then

- arbitrary code: requires $\binom{m}{2} = \Theta(m^2)$ comparison operations;

- linear code: requires $m - 1$ weight operations.

One of the interests of linear codes is that their algebraic structure allows us to speed up typical operations that are performed in a code. We will see more examples in this chapter.

6.1 Basis, generator matrices and normal forms

Since a linear code is a subspace, we can describe it by the *basis* that generates it.

Definition 6.6 (Basis)**.** A *basis* of a linear $[n, k]$-code C is set of k vectors $c_1, \ldots, c_k \in C$ such that for any vector $c \in C$ there exist $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_2$ with

$$c = \lambda_1 c_1 + \cdots + \lambda_k c_k. \tag{6.1}$$

Lemma 6.7. *Every linear $[n, k]$-code C satisfies $|C| = 2^k$.*

Proof. There are 2^k choices for $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_2$, so $|C| \leq 2^k$. Moreover, different choices give rise to different codewords, concluding that $|C| = 2^k$. Indeed, suppose not, then there exist $\lambda_1, \ldots, \lambda_k, \mu_1, \ldots, \mu_k \in \mathbb{F}_2$, not all equal (that is, there exists $i \in [k]$ such that $\lambda_i \neq \mu_i$) with

$$\lambda_1 c_1 + \cdots + \lambda_k c_k = \mu_1 c_1 + \cdots + \mu_k c_k \ ,$$

or, equivalently,

$$(\lambda_1 - \mu_1) c_1 + \cdots + (\lambda_k - \mu_k) c_k = 0 \ .$$

Since $\lambda_i - \mu_i \neq 0$ we should have $\lambda_i - \mu_i = 1$. Using that $-1 = 1$ in $\mathbb{F}_2$, we can write,

$$c_i = (\lambda_i - \mu_i) c_i = (\lambda_1 - \mu_1) c_1 + \cdots + (\lambda_{i-1} - \mu_{i-1}) c_{i-1} + (\lambda_{i+1} - \mu_{i+1}) c_{i+1} + \cdots + (\lambda_k - \mu_k) c_k$$

Thus, c_i can be expressed as a linear combination of the other codewords, which contradicts the fact that $c_1, \ldots, c_k$ is a basis of a subspace of dimension k. □

Algorithmic Remark. The algebraic structure of linear codes allows us to store them in a much cheaper way. For a code of size 2^k,

- arbitrary code: we need to store each codeword, using a total of $n2^k$ bits.
- linear code: it suffices to store a basis of the code, that has size k, using a total of nk bits.

Example 6.8. Let C_1 and C_2 be the codes in Examples 6.3 and 6.4. The code C_1 is generated by the basis $\{(1000\dots01), (0100\dots01), \dots, (0000\dots011)\}$, and thus its dimension/rank is $k = n-1$ and $|C_1| = 2^{n-1}$. The basis is not unique, another possible basis is $\{(1100\dots00), (0110\dots00), \dots, (0000\dots011)\}$

The code C_2 is generated by the basis $\{(11000\dots0), (00110\dots0), \dots, (00\dots011)\}$, and thus its rank is $k = n/2$ and $|C_2| = 2^{n/2}$.

Definition 6.9 (Generator Matrix)**.** Let C be a linear $[n, k]$-code. Given an ordered basis $c_1, \dots, c_k$ of C, the associated *generator matrix* of C is a $k \times n$ matrix whose rows correspond to $c_1, \dots, c_k$; that is, if $c_i = c_{i1}c_{i2}\dots c_{in}$, for $i \in [k]$, then

$$B = \begin{pmatrix} c_{11} & \dots & c_{1j} & \dots & c_{1n} \\ \vdots & & \vdots & & \vdots \\ c_{i1} & \dots & c_{ij} & \dots & c_{in} \\ \vdots & & \vdots & & \vdots \\ c_{k1} & \dots & c_{kj} & \dots & c_{kn} \end{pmatrix}$$

If $\lambda = (\lambda_1, \dots, \lambda_k) \in \mathbb{F}_2^k$, the matrix form of (6.1) is

$$c = \lambda B.$$

Remark. A linear code C has many bases. Each ordered basis gives rise to a different generator matrix. A generator matrix can be obtained from another generator matrix by performing elementary row-operations (columns-operations are not allowed!).

Example 6.10. Let C_1 be the code from Example 6.3. The generator matrix given by the basis chosen in Example 6.8 is

$$B = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \end{pmatrix}$$

An alternative basis of C_1 is $\{(100\dots01), (010\dots01), \dots, (0000\dots11)\}$, which gives rise to another generator matrix of C_1

$$B = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 1 \\ 0 & 1 & 0 & \dots & 0 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \end{pmatrix}$$

Definition 6.11 (Normal form)**.** Given a linear $[n, k]$ code C, a generator matrix B_0 is a *normal form* if it has the form $B_0 = (I_k|A)$, where I_k is the $k \times k$ identity matrix.

Recall the construction of the Hamming $[7,4]$-code described in Exercise 3 from Problem Sheet 3.

Example 6.12. Let C be the Hamming $[7,4]$-code, which is linear. It is easy to see that it has generator matrix

$$\left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

Given a code, one can obtain another code by permuting the same positions in every vector.

Definition 6.13 (Equivalent Codes)**.** Two linear codes C_1 and C_2 are *equivalent* if there exist a generator matrix B_1 of C_1 and a generator matrix B_2 of C_2 such that B_1 can be obtained from B_2 by a permutation of the columns.

Two equivalent codes have the same length, dimension, minimum distance and size. Not all linear codes have normal forms, but there always exists an equivalent code that admits a normal form.

Theorem 6.14. *For every linear $[n,k]$-code C, there exist an equivalent linear $[n,k]$-code C_0 that has a normal form.*

Sketch proof. Let B be a generator matrix of C; this can be transformed into a matrix of the form $(I_k|A)$ by performing elementary row operations (permutation, multiplication by a non-zero scalar, addition) and permutations of columns. Row operations do not change the code (only its basis) and column permutations give rise to an equivalent code. □

6.2 Parity Check Matrix

Definition 6.15 (Parity check matrix)**.** Let C be a linear $[n,k]$-code. A *parity check matrix* H of C is an $(n-k)\times n$ matrix such that

$$C = \{x \in \mathbb{F}_2^n : \; Hx^T = 0\} \; .$$

Remark. Parity check matrices provide us a quick way of deciding whether a string is in a linear code.

The following result gives us a tool to compute the parity check matrix of a code using its normal formal.

Theorem 6.16. *Let C be a linear $[n,k]$-code and suppose C has a normal form $B_0 = (I_k|A)$. Then,*

$$H = (A^T|I_{n-k}) \; ,$$

is a parity check matrix of C.

Proof. To prove that H is a parity check matrix of C we need to show that $Hx^T = 0$ if and only if $x \in C$.

If $x \in C$, then there exists $\lambda \in \mathbb{F}_2^k$ such that $x = \lambda B_0 = (\lambda|\lambda A)$. Thus,

$$Hx^T = (A^T|I_{n-k})(\lambda|\lambda A)^T = A^T\lambda^T + (\lambda A)^T = (\lambda A)^T + (\lambda A)^T = 0.$$

If $Hx^T = 0$ and writing $x = (\lambda|\mu)$, where $\lambda \in \mathbb{F}_2^k$ corresponds to the first k positions of x and $\mu \in \mathbb{F}_2^{n-k}$ corresponds to the last $n-k$ ones, then,

$$0 = Hx^T = (A^T|I_{n-k})(\lambda|\mu)^T = A^T\lambda^T + \mu^T = (\lambda A)^T + \mu^T \ ,$$

which implies $\mu = \lambda A$. It follows that,

$$x = (\lambda|\mu) = (\lambda|\lambda A) = \lambda(I_k|A) = \lambda B_0 \ ,$$

and $x \in C$, since it can be generated by the basis. □

Example 6.17. The parity check matrix of the Hamming $[7,4]$-code in Example 6.12 is

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

The parity check matrix also allows us to quickly determine the minimum distance of the code.

Theorem 6.18. *Let C be a linear $[n,k]$-code with parity check matrix H. If r is the smallest integer for which there are r linearly dependent columns in H, then $d_H(C) = r$.*

Proof. Recall that H is a $(n-k) \times n$ matrix and let $h_1, \dots, h_n$ be its columns. Consider a collection of r of these columns $h_{i_1}, \dots, h_{i_r}$ that are linearly dependent. Since r is the smallest integer for which this is possible, we have,

$$h_{i_1} + \cdots + h_{i_r} = 0 \ . \tag{6.2}$$

Let $c \in \mathbb{F}_2^n$ be the vector that has 1s at positions $i_1, \dots, i_r$ and 0s everywhere else; its weight is $w(c) = r$. The equation (6.2) can be written as $Hc^T = 0$ and $c \in C$. By Proposition 6.5, we have

$$d_H(C) \leq w(c) = r \ .$$

Now, let $\hat{c} \in C$ such that $w(\hat{c}) = d_H(C)$, which exists by Proposition 6.5, and note that $H(\hat{c})^T = 0$. So, the $w(\hat{c}) = d_H(C)$ columns of H which correspond to positions where $\hat{c}$ has 1s, are linearly dependent, since $H(\hat{c})^T$ is their sum. We conclude that $d_H(C) \geq r$. □

Example 6.19. Consider again the Hamming $[7,4]$-code with parity check matrix

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

We know that $d_H(C) = 3$ but we will compute it using Theorem 6.18. Let r be the smallest integer such that there are r linearly dependent columns in H. Clearly, $r \leq 3$; for instance $h_2 + h_5 + h_6 = 0$. Moreover, since all the columns in H are different, $r \geq 3$. By Theorem 6.18, it follows that $d_H(C) = 3$.

6.3 Hamming codes

We have already seen the Hamming $[7,4]$-code as an example of linear code. In this section we describe a family of codes that were introduced by Richard Hamming in 1950.

Definition 6.20 (Hamming code)**.** Let $r \in \mathbb{N}$. The *Hamming code of order* r is a linear code whose parity check matrix H has as its columns all $2^r - 1$ non-zero vectors of length r.

The Hamming $[7,4]$-code is equivalent to the Hamming code of order 3. In Example 6.17 we have obtained its parity check matrix. Note that all the non-zero vectors of length 3 appear in its columns,

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

The following proposition gives the basic features of a Hamming code.

Proposition 6.21. *The Hamming code of order r is a linear $[n,k]$-code C with*

i) $n = 2^r - 1$ *and* $k = 2^r - r - 1$;

ii) $|C| = 2^{2^r - r - 1}$ *and* $R(C) = 1 - \frac{r}{2^r - 1}$;

iii) $d_H(C) = 3$.

Proof. By definition of Hamming code, its parity check matrix H has r rows and $2^r - 1$ columns. Since H has size $(n-k) \times n$, where n is the length and k the rank of C, one has $n = 2^r - 1$ and $k = 2^r - r - 1$. Moreover, a linear code of rank k has size 2^k, whereby $|C| = 2^{2^r - r - 1}$ and

$$R(C) = \frac{k}{n} = \frac{2^r - r - 1}{2^r - 1} = 1 - \frac{r}{2^r - 1} \ .$$

To compute the minimum distance one can use Theorem 6.18. If we take any two different columns of the parity check matrix, then their sum is equal to another column. Thus, these three vectors are linearly dependent. However, any two distinct columns are not linearly dependent as their sum is non-zero. Hence, by Theorem 6.18, the minimum distance of C is 3. □

Remark. The transmission rate of a Hamming code C of order r tends to 1 when $r \to \infty$. However, since $d_H(C) = 3$, Hamming codes can only correct 1 error. Thus, Hamming codes are very efficient but are not very robust in the transmission of data, and they are only used in applications where errors are unlikely to occur. For instance, Hamming codes are used to store data in hard drives, since errors/erasures are extremely unusual.

Hamming codes will serve us as a first example of decoding scheme for linear codes. We will first prove the following proposition.

Proposition 6.22. *Let C be the Hamming code of order r with parity check matrix H. For every $x \in \mathbb{F}_2^n$, there exists a unique $c \in C$ such that $d_H(x,c) \le 1$.*

Moreover, if $x \notin C$ and i is the position where x and c differ, then

$$Hx^T = h_i \ ,$$

where h_i is the i-th column of H.

Proof. If $x \in C$, then we can set $c = x$ and $d_H(c, x) = 0 \leq 1$. So we may assume that $x \notin C$. By definition of the parity check matrix, we have $Hx^T = s \in \mathbb{F}_2^r$, with $s \neq 0$. Since H contains as columns all non-zero vectors in $\mathbb{F}_2^r$, there exists $i \in [n]$, such that $s = h_i$. Let e_i be the vector that has a single 1 at the i-th position and 0s everywhere else. Using that $h_i = He_i^T$, we can write

$$0 = h_i + h_i = s + h_i = Hx^T + He_i^T = H(x + e_i)^T \ ,$$

and $x + e_i \in C$. Setting $c = x + e_i$ and using Lemma 6.2, we conclude the first part

$$d_H(x, c) = d_H(x, x + e_i) = w(x + x + e_i) = w(e_i) = 1 \ .$$

The second part follows directly,

$$Hx^T = H(c + e_i)^T = Hc^T + He_i^T = 0 + h_i = h_i \ .$$

□

The previous proposition suggests a decoding scheme.

Hamming decoding scheme for $x \in \mathbb{F}_2^n$:

1. Calculate $s = Hx^T$;
2. Decode it:
   - 2a. if $s = 0$, then $x \in C$, so return x;
   - 2b. if $s \neq 0$, by Proposition 6.22 there exists i such that $s = h_i$, so return $x + e_i$.

The above algorithm is a particular case of minimum distance decoding and can correct up to 1 error. Namely, if $c \in C$ is transmitted, 1 error occurs at the i-th position and x is received, then the above algorithm will correct this. In the next section we will extend this algorithm for arbitrary linear codes.

Example 6.23. Let C be the Hamming code of order 3 presented in Example 6.12. If $x = (0110011)$ is received, then

$$\left(\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \cdot \left(\begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \\ \hline 0 \\ 1 \\ 1 \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ 0 \end{array}\right) \ .$$

This implies that $x \in C$ and it is decoded as x.

If $x = (0111011)$ is received, then

$$\left(\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \cdot \left(\begin{array}{c} 0 \\ 1 \\ 1 \\ 1 \\ \hline 0 \\ 1 \\ 1 \end{array}\right) = \left(\begin{array}{c} 0 \\ 1 \\ 1 \end{array}\right) =: s \ .$$

Thus, s equals the 4th column of H. Therefore, it is decoded to $x + e_4 = (0110011)$, which belongs to C, as we have checked before.

6.4 Syndrome decoding

The same idea we have used to decode the Hamming code, can be used for other linear codes.

Definition 6.24. (Syndrome) Let C be a linear $[n, k]$-code with parity check matrix H. For any vector $s \in \mathbb{F}_2^{n-k}$, we let the *coset* of s be defined as

$$Q(s) := \{x \in \mathbb{F}_2^n \mid Hx^T = s\} .$$

If $x \in Q(s)$, then we say that s is the *syndrome* of x.

Remark. The following properties hold:

1. By definition of H, $Q(0) = C$.
2. For any two $x, y \in Q(s)$, we have $x + y \in C$, since
$$H(x + y)^T = Hx^T + Hy^T = s + s = 0 .$$
3. The family of subsets $\{Q(s) : s \in \mathbb{F}_2^{n-k}\}$ is an equipartition of $\mathbb{F}_2^n$ (disjoint sets of the same size).

Definition 6.25 (Coset leader)**.** For every $s \in \mathbb{F}_2^{n-k}$, a *coset leader* $q(s)$ is a vector of $Q(s)$ with smallest weight; that is, $q(s)$ satisfies

$$w(q(s)) = \min_{x \in Q(s)} w(x) .$$

Note that coset leaders may not be unique.

Example 6.26. Let C be the linear $[4, 1]$-code with parity check matrix

$$H = \left(\begin{array}{c|ccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right)$$

The coset leader of $s = (100)$ is (0100). However, the coset leader of $s = (110)$ can be chosen to be either (0110) or (1001), which have the same weight.

One can use the coset leaders to implement a minimum distance decoding.

Theorem 6.27. *Let C be a linear code. If $x \in Q(s)$ and $q(s)$ is a coset leader of $Q(s)$, then $c = x + q(s) \in C$ and c is a nearest codeword to x.*

Proof. Since x, $q(s) \in Q(s)$, it follows from the previous remark that $c \in C$.

We will prove the second part by contradiction. Assume that there exists $c' \in C$ with $d_H(c', x) < d_H(c, x)$. By Lemma 6.2 we have

$$w(c' + x) = d_H(c', x) < d_H(c, x) = w(c + x) = w(x + q(s) + x) = w(q(s)) .$$

Since $c' \in Q(0)$, we have

$$H(c' + x)^T = H(c')^T + Hx^T = 0 + s = s ,$$

and $c' + x \in Q(s)$. This leads to a contradiction since $q(s)$ is a vector of smallest weight in $Q(s)$. □

Theorem 6.27 suggests the following decoding scheme.

Syndrome decoding scheme for $x \in \mathbb{F}_2^n$:

1. Calculate $s = Hx^T$.
2. Choose a coset leader $q(s)$.
3. Decode x as $x + q(s)$.

By Theorem 6.27, this is a minimum distance decoder.

Example 6.28. Let C be the code in Example 6.26 with parity check matrix

$$H = \left(\begin{array}{c|ccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array} \right)$$

We first list all the coset leaders of C

syndrome	(000)	(100)	(010)	(001)	(110)	(101)	(011)	(111)
coset leaders	(0000)	(0100)	(0010)	(0001)	(0110) (1001)	(1010) (0101)	(0011) (1100)	(1000)

We show now how to decode words using syndrome decoding:

- Let $x = (1111)$. Its syndrome is $Hx^T = (000)$, and it is decoded to $(1111)+(0000) = (1111)$. In fact, $x \in C$.
- Let $x = (1110)$. Its syndrome is $Hx^T = (001)$, and it is decoded to $(1110)+(0001) = (1111)$.
- Let $x = (1010)$. Its syndrome is $Hx^T = (101)$, and it can be decoded either to $(1010) + (1010) = (0000)$ or to $(1010) + (0101) = (1111)$.

6.5 The Gilbert-Varshamov bound

We will use linear codes to derive a last lower bound on $A(n,d)$ that improves on the bound given in Proposition 4.4.

Theorem 6.29 (The Gilbert-Varshamov bound)**.** *Let $2 \le d \le n$ and let k be the largest integer that satisfies $2^k < \frac{2^n}{b^{n-1}_{d-2}}$. Then*

$$A(n,d) \ge 2^k \ .$$

Proof. We will construct a linear $[n,k]$-code C with $d_H(C) \ge d$ by constructing a parity check matrix H of size $(n-k) \times n$ whose minimum collection of linearly dependent columns has size at least d. Lemma 6.7 implies that $|C| = 2^k$ and Theorem 6.18 implies that $d_H(C) \ge d$. If so, we will conclude,

$$A(n,d) \ge |C| = 2^k \ .$$

We will construct H by selecting its columns *greedily*, so that no column is the sum of up to $d-2$ columns that have been previously selected. This implies no $d-1$ or fewer columns are linearly dependent, as desired.

Assume that we have already selected ℓ columns that satisfy this. The $(\ell+1)$-th column could be any vector of length $n-k$, except for a sum of up to $d-2$ vectors among the chosen ℓ. The number of forbidden options is at most

$$\sum_{m=0}^{d-2} \binom{\ell}{m} .$$

Since there are 2^{n-k} possible vectors of length $n-k$, as long as $2^{n-k} > \sum_{m=0}^{d-2} \binom{\ell}{m}$, we can select an $(\ell+1)$-th column for H.

In order for H to be a parity check matrix it should have n columns. The algorithm will reach the n-th step ($n = \ell + 1$) if

$$2^{n-k} > \sum_{k=0}^{d-2} \binom{n-1}{k} = b_{d-2}^{n-1} ,$$

where the last equality follows from Lemma 4.2, or equivalently

$$2^k < \frac{2^n}{b_{d-2}^{n-1}} .$$

This is satisfied by the hypothesis of the theorem, so we are done. □

IMPORTANT CONCEPTS OF THIS CHAPTER

- Linear codes are subspaces of $\mathbb{F}_2^n$. Thus, they have bases, generator matrices and can be expressed as the null space of a matrix (parity check matrix).
- A parity check matrix can be easily computed from the normal form of a code.
- Linear codes have many positive aspects: they provide good lower bounds for $A(n,d)$ and they can perform basic code operations (compute minimum distance, store, check if $x \in C$, decode) in a more efficient way.
- The minimum distance can be easily computed for linear codes (Proposition 6.5 and Theorem 6.18).
- Hamming codes are an infinite family of codes, which are efficient and easy to decode, but not very robust.
- Syndrome decoding gives a minimum distance decoding for all linear codes.