University of Cambridge

Part II of the Mathematical Tripos

Coding and Cryptography

Lectured by Rachel Camina, Lent 2024–25

Notes by Avish Kumar
ak2461@cam.ac.uk
https://ak1089.github.io/maths/notes

Version 1.7

Contents

1 Noiseless Coding

1.1 The Coding Problem

The general problem of coding is that of transmitting a message across a communication channel. For example, if we wish to send an email containing a message $m =$ "Call me!", we may encode this as a sequence of binary strings using the standard ASCII format.

Under this code, $f(\text{"}C\text{"}) = \texttt{1000011}$. In fact, each character is mapped to seven binary digits, or bits. The entire message is the concatenation of these strings: "Call me!" becomes:

$$f^*(\text{"Call me!"}) = \texttt{10000111100001110110011011000100000110110111001010100001}.$$

Definition 1.1 (Source, Encoder, Channel, Receiver, Decoder)

More generally, we have a *source*, often called Alice. She uses an *encoder* to convert plaintext messages into encoded messages. These encoded messages are sent through a *channel*: this channel may be *noisy*, and introduce errors into the code. The encoded message is received by a *receiver* Bob, who uses a *decoder* to convert it back into the original plaintext.

Given a source and a channel, described probabilistically, we want to design an encoder and decoder in order to transmit source information across the channel. We might want certain properties:

1. Economy: we would like to minimise the amount of unnecessary information sent: the code should not be too long, as it wastes time and money.
2. Reliability: the decoder should be able to successfully decipher the plaintext with very high probability, or mistakes should be detectable.
3. Privacy: we may want only someone with the decoder to be able to read the message.

Accomplishing this last desideratum is the aim of *cryptography* in particular, while *coding* deals with the first two. How might we achieve these?

Remark 1.2 (Economy and Reliability)

Morse code is economic in that it attempts to minimise message length. This is done by giving the shorter codes to letters which are used more frequently. For example, $E = \cdot$, while $Q = - - \cdot - -$.

The ISBN system for numbering books is reliable. Each book has a unique ten-digit ISBN: the first nine digits encode information about the book (its publisher, ID, and region) while the last digit is a *check digit* chosen such that $10a_1 + 9a_2 + \cdots + 2a_9 + a_{10} \equiv 0 \pmod{11}$. This has robust error-detection capabilities. For example, a transposition of any two digits a_i and a_j will add $(j-i)(a_j - a_i) \not\equiv 0 \pmod{11}$ to the left hand side. This means a single error can be detected, since the congruence will be broken.

1.2 Communication Channels

Definition 1.3 (Channel)

A communication channel takes letters from an input alphabet $\Sigma_1 = \{a_1, \ldots, a_r\}$ and emits letters from an output alphabet $\Sigma_2 = \{b_1, \ldots, b_s\}$. It is determined by the probabilities

$$\mathbb{P}[y_1 \ldots y_k \text{ emitted} \mid x_1 \ldots x_k \text{ input}] \text{ where } x_i \in \Sigma_1^*, y_i \in \Sigma_2^*.$$

Note: The important feature of a channel is that it is not necessarily perfect! Much like in the real world, where we deal with problems like TV static or data corruption, the channels we will study are affected by noise.

Definition 1.4 (Discrete Memoryless Channel)

A discrete memoryless channel over a finite alphabet is a channel for which the probabilities

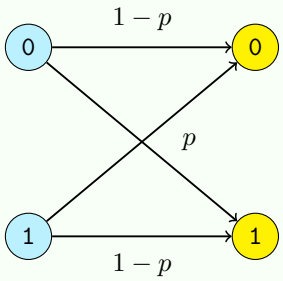
$$P_{ij} = \mathbb{P}[b_j \text{ received} \mid a_i \text{ sent}]$$

are the same every time the channel is used, independent of past and future channel use. This is the *memoryless property*, while the discrete nature is given by the alphabets.

We often identify the channel with its *channel matrix* P, which is the $r \times s$ matrix with entries p_{ij} equal to those probabilities. Note that the rows of P, but not necessarily its columns, sum to 1: we say that P is a *stochastic matrix*.

Example 1.5 (Binary Symmetric/Erasure Channel)

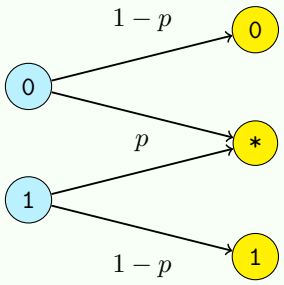
For example, a Binary Symmetric Channel with probability $0 \leqslant p \leqslant 1$ of error is a DMC over the binary alphabet $\Sigma_1 = \Sigma_2 = \{0, 1\}$. In particular, any bit sent has a probability p of being flipped by the channel due to noise. This can be seen in the below diagram.



$$P = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$$

Usually, we assume $p < 0.5$. If $p > 0.5$, then we can just pre-flip the bits sent to reduce the error probability (since any bit is likely flipped back by the channel). If $p = 0.5$, then every single bit received is equally likely to be 0 and 1, independently of what was actually transmitted, so the channel is entirely useless (pure noise).

A Binary Erasure Channel is similar, taking $\Sigma_1 = \{0, 1\}$ but $\Sigma_2 = \{0, 1, *\}$, with the $*$ understood to be an *erasure*. Each bit transmitted has a probability $0 \leqslant p \leqslant 1$ of being erased, making it unreadable, and is transmitted correctly otherwise (never flipped), giving:



$$P = \begin{pmatrix} 1-p & p & 0 \\ 0 & p & 1-p \end{pmatrix}$$

where the columns correspond to 0, $*$, and then 1.

Definition 1.6 (Capacity)

The *capacity* of a channel is the highest rate at which information can be reliably transmitted over the channel. Here, the rate is measured as units of information per unit time: for a binary channel, this might be the number of decoded bits per transmitted bit. High reliability is achieved by an arbitrarily low probability of error.

1.3 Strings and Alphabets

We frequently work with alphabets, which are simply sets of elements called letters or characters. These are the building blocks of a language: the input alphabet of a code is the set of atoms which are encoded into something else.

Definition 1.7 (String, Concatenation, Length)

For an alphabet Σ, we define the set of Σ-strings to be $\Sigma^* = \bigcup_{n \geqslant 0} \Sigma^n$. These are usually written as concatenations rather than tuples, so that the set of binary alphabet strings is

$$\Sigma_{01}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, \ldots\}$$

The *length* of a string is the number of letters contained. Here ε is the empty string with length 0. If $x = x_1 x_2 \ldots x_r$ and $y = y_1 y_2 \ldots y_s$ are Σ-strings, their concatenation is given by $xy = x_1 \ldots x_r y_1 \ldots y_s$.

For two alphabets Σ_1 and Σ_2, a *code* is a function $f : \Sigma_1 \to \Sigma_2^*$. The strings $\{f(x) : x \in \Sigma_1\}$, or the image of f, are called codewords.

Example 1.8 (Polybius Square)

For example, the Polybius Square is a cipher developed by Ancient Greek polymath Polybius, who created a way to encode Greek as numbers.

The input alphabet Σ_1 was the 24 Greek letters α to ω, while the output alphabet Σ_2 was the set $\{1, 2, 3, 4, 5\}$. Each letter was mapped to precisely two digits from 1 to 5 for easy transmission (using every pair except 55). This made the codewords the set

$$\{1 \ldots 5\}^2 = \{11, 12, 13, 14, 15, 21, 22, \ldots 52, 53, 54\}.$$

Note: For English-language codes, we do not necessarily have Σ_1 being $\{\mathrm{a} \ldots \mathrm{z}\}$ the set of letters. The domain of the code function is the set of atoms of the code, which is often pairs of letters, or even more commonly entire words.

We apply a code by encoding $x_1 x_2 \ldots x_n \in \Sigma_1^*$ as $f(x_1) f(x_2) \ldots f(x_n) \in \Sigma_2^*$. This extends f the code function from atoms to entire words in the input language, which we call $f^* : \Sigma_1^* \to \Sigma_2^*$.

However, not every function $f : \Sigma_1 \to \Sigma_2^*$ works as a code.

Definition 1.9 (Decipherable)

A code f is *decipherable* if f^* is injective, so that every string in Σ_2^* arises from at most one message. Without this condition, the output of encoding might have come from multiple possible inputs, and we would have no way of knowing which when decoding it.

Proposition 1.10 (Decipherability requires injectivity)

A decipherable code f requires f injective. However, this is not a sufficient condition.

Proof: Firstly, if f is not injective, then $f(x) = f(y)$ where $x \neq y \in \Sigma_1$. But then the encoding of x and y when treated as members of Σ_1^* is the same, violating injectivity of f^*.

However, this is not a sufficient condition. Suppose $\Sigma_1 = \{1, 2, 3, 4\}$ and $\Sigma_2 = \{\texttt{0}, \texttt{1}\}$. Define

$$f(1) = \texttt{0} \qquad f(2) = \texttt{1} \qquad f(3) = \texttt{00} \qquad f(4) = \texttt{01}$$

so that f is injective, but $f^*(1112) = \texttt{0001} = f^*(34)$, so f^* is not. □

How do we construct decipherable codes? There are a few basic properties of codes which guarantee decipherability (none of these are necessary, but are all sufficient) provided that f is injective.

1. A *block code* is a code where all codewords are of the same length. For example, the Polybius cipher had all codewords of length 2, and so it can be decoded by considering the output as a list of length-2 strings.
2. A *comma code* reserves one letter in Σ_2 to act as the comma, which appears at the end of every output of f and nowhere else. It thus delimits words in Σ_2^*, so we know where each letter in the original input to the code was mapped.
3. A *prefix-free* (or *instantaneous*) code is a code where no codeword is a prefix of any other codeword: for any $x, y \in \Sigma_1$, we have $f(x) \neq f(y)\alpha$ for any $\alpha \neq \varepsilon \in \Sigma_2^*$.

Note: In fact, block codes and comma codes are special cases of prefix-free codes.

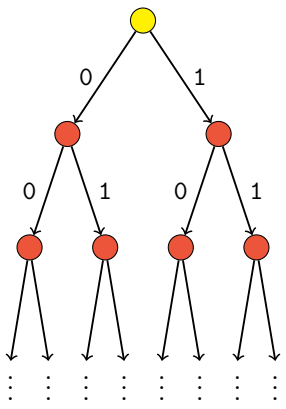
Theorem 1.11 (Kraft's Inequality)

Let $\Sigma_1 = \{x_1, \dots x_m\}$ and $|\Sigma_2| = a$. Then a prefix-free code $f : \Sigma_1 \to \Sigma_2^*$ with word lengths $s_1 \dots s_m$ (where $|f(x_i)| = s_i$) exists if and only if

$$\sum_{i=1}^{m} a^{-s_i} \leqslant 1$$

Proof: ($\Rightarrow$) Consider an infinite tree where each node has a descendents corresponding to the a letters of Σ_2. Then each codeword corresponds to precisely one of these nodes, where the path to the node spells out the codeword along the branches taken.

Assuming f is prefix-free, no codeword is the ancestor of any other. View the tree as a network, with water being pumped in at the root, where each node divides the flow equally between each descendant. The total amount of water extracted at the codewords is therefore the sum of a^{-s_i}, which is at most the total amount of water pumped in, demonstrating the inequality.



($\Leftarrow$) Conversely, we can construct a prefix-free code with word lengths $s_1 < s_2 < \cdots < s_m$. Choose codewords sequentially, ensuring that any previous codewords are not prefixes. Suppose that the r^{th} codeword has no valid code available. Then constructing the tree above gives

$$\sum_{i=1}^{r-1} a^{-s_i} = 1 \implies \sum_{i=1}^{m} a^{-s_i} > 1$$

which contradicts our assumption. □

Theorem 1.12 (McMillan / Karush)

Every decipherable code satisfies Kraft's inequality.

Proof: Let $f : \Sigma_1 \to \Sigma_2^*$ be a decipherable code with word lengths $s_1 \dots s_m$, where $s = \max s_i$. For any $r \in \mathbb{N}$, we must have

$$\left(\sum_{i=1}^{m} a^{-s_i}\right)^r = \left(\sum_{\ell=1}^{rs} b_\ell a^{-\ell}\right)$$

where b_ℓ is the number of ways of choosing r codewords with total length ℓ. Since f is decipherable, we know that $b_\ell \leqslant |\Sigma_2|^\ell = a^\ell$, since no string can be the encoding of more than one set of codewords. This means that we can write

$$\left(\sum_{i=1}^{m} a^{-s_i}\right)^r \leqslant \left(\sum_{\ell=1}^{rs} a^\ell a^{-\ell}\right) = rs \implies \left(\sum_{i=1}^{m} a^{-s_i}\right) \leqslant (rs)^{1/r}$$

But this is true for any r, and $(rs)^{1/r} \to 1$ as $r \to \infty$. Therefore the left hand side of the inequality is at most 1, which is exactly the statement of Kraft's inequality. □

As a result, we mostly restrict our attention to prefix-free codes.

1.4 Mathematical Entropy

Entropy is a measure of "randomness" or "uncertainty". Suppose a random variable X takes values $x_1 \dots x_n$ with probabilities $p_1 \dots p_n$, where we have $0p_i \leqslant 1$ for all i and $\sum p_i = 1$. Loosely, the *entropy* $H(X)$ is the expected number of yes/no questions required to determine the value of X.

This is not a formal definition yet: we consider some examples first.

Example 1.13 (Basic Entropy Examples)

Let's consider X taking values $x_1 \dots x_4$.

If $p_1 = p_2 = p_3 = p_4 = 1/4$, then asking precisely two yes/no questions can consistently determine the value of X. For example, the two questions could be "is $X \in \{x_1, x_2\}$?" and "is $X \in \{x_1, x_3\}$?". Here, this means $H(X) = 2$ directly.

Now, suppose $(p_1, p_2, p_3, p_4) = (1/2, 1/4, 1/8, 1/8)$. Then we could ask the question "$X = x_1$?" and finish with one question half the time. If the answer is no, we ask "$X = x_2$?" and again be done half the time (so a quarter overall). Failing that, we ask "$X = x_3$?" and know the value of X with certainty after three questions.

This gives $H(X) = 1 \times 1/2 + 2 \times 1/4 + 3 \times 1/8 + 3 \times 1/8 = 7/4 < 2$, and so the first random variable is "more random". This aligns with our intuition.

This gives us enough to write down our formal definition.

Definition 1.14 (Entropy)

For a random variable X taking values $x_1 \dots x_n$ with probabilities $p_1 \dots p_n$, where we have $0 \leqslant p_i \leqslant 1$ for all i and $\sum p_i = 1$, the *entropy* $H(X)$ is defined to be

$$H(X) = H(p_1, \dots, p_n) = -\sum_{i=1}^{n} p_i \log p_i.$$

Note: In this course, we always consider the logarithm to be defined as $\log_2$ (the logarithm with base 2), rather than the natural logarithm e, due to our focus on binary.

Note: This definition breaks down if $p_i = 0$ for some i: in this case, we take $p_i \log p_i = 0$ as a convention, since we could have excluded p_i and x_i for an equivalent distribution.

Corollary: As $p_i \log p_i \leqslant 0$ for $0 \leqslant p_i \leqslant 1$, the entropy $H(X) \geqslant 0$.

Example 1.15 (Entropy of a Biased Coin)

Toss a biased coin which lands heads with probability p and tails with probability $1-p$. Then

$$h(p) = H(p, 1-p) = -p \log p - (1-p) \log(1-p)$$

Plotting this, we get an arch-shaped curve with $h(0) = h(1) = 0$, since the outcome is certain and thus there is no randomness. The graph is symmetric, which makes sense, and we get a peak at $p = 1/2$, which is the case for a fair coin (which is therefore maximal entropy).

We now prove a result which will come up frequently in the study of entropy.

Theorem 1.16 (Gibbs' Inequality)

Let $\mathbf{p} = (p_1 \dots p_n)$ and $\mathbf{q} = (q_1 \dots q_n)$ be probability distributions. Then

$$-\sum_{i=1}^{n} p_i \log p_i \leqslant -\sum_{i=1}^{n} p_i \log q_i.$$

with equality if and only if $\mathbf{p} = \mathbf{q}$.

Proof: Since $\log x = \ln(x)/\ln(2)$, we may prove the equality using ln in place of log, and dividing through both sides afterwards. Note that $\ln x \leqslant x - 1$ with equality if and only if $x = 1$.

Let $I = \{1 \leqslant i \leqslant n : p_i > 0\}$ be the set of nontrivial indices. Then

$$\ln(q_i/p_i) \leqslant q_i/p_i - 1 \quad \forall\, i \in I.$$

and therefore we have

$$\sum_{i \in I} p_i \ln(q_i/p_i) \leqslant \sum_{i \in I} q_i - \sum_{i \in I} p_i = \sum_{i \in I} q_i - 1 \leqslant 0$$

Rearranging this inequality yields

$$-\sum_{i \in I} p_i \ln p_i \leqslant -\sum_{i=1}^{n} p_i \ln p_i \leqslant -\sum_{i=1}^{n} p_i \ln q_i$$

as required. Equality is only possible if we had equality in the first line, with all $i \in I$ satisfying $\ln(q_i/p_i) = q_i/p_i - 1 \implies q_i/p_i = 1 \implies q_i = p_i$ as desired, and thus the proof holds. □

Corollary: $H(p_1 \dots p_n) \leqslant \log n$ with equality if and only if $p_i = 1/n$ for all i.

1.5 Optimal Noiseless Coding

Recall that the coding problem considers alphabets Σ_1 and Σ_2 of sizes $m, a \geqslant 2$ respectively. When considering a channel, we model the source as a sequence of random variables $X_1, X_2 \dots$ which take values in Σ_1.

Definition 1.17 (Memoryless Source)

A *Bernoulli* (or *memoryless*) source is a sequence $X_1, X_2, \dots$ of independently and identically distributed random variables.

Definition 1.18 (Expected Word Length, Optimal Code)

Consider a memoryless source. Let $\Sigma_1 = \{\mu_1 \dots \mu_m\}$ and define $p_i = \mathbb{P}[X_1 = \mu_i]$. The expected word length S of a code $f : \Sigma_1 \to \Sigma_2^*$ with word length $s_1 \dots s_m$ is therefore

$$\mathbb{E}[S] = \sum_{i=1}^{m} p_i s_i$$

The code f is then said to be *optimal* if it has the shortest possible expected word length among decipherable codes: that is, if it minimises $\mathbb{E}[S]$.

This brings us to one of the most important results in information theory.

Theorem 1.19 (Shannon's Noiseless Coding Theorem)

The expected word length of an optimal decipherable code $f : \Sigma_1 \to \Sigma_2^*$ satisfies

$$\frac{H(X)}{\log(a)} \leqslant \mathbb{E}[S] < \frac{H(X)}{\log(a)} + 1.$$

This theorem was proved in 1948 by Claude Shannon, the father of information theory. It is also known by several other names, like *Shannon's Source Coding Theorem for Symbol Codes.*

Proof: The lower bound is given by combining Gibbs' and Kraft's inequalities (1.16 and 1.11), taking $q_i = a^{-s_i}/c$, where $c = \sum a^{-s_i} \leqslant 1$ is such that $\sum q_i = 1$. Then

$$\begin{aligned}
H(X) &= -\sum_{i=1}^{m} p_i \log p_i \\
&\leqslant -\sum_{i=1}^{m} p_i \log q_i \qquad \text{(by Gibbs')} \\
&= -\sum_{i=1}^{m} p_i \log(a^{-s_i}/c) \\
&= \log a \sum_{i=1}^{m} p_i s_i + \sum_{i=1}^{m} p_i \log c \\
&= \mathbb{E}[S] \times \log a + \log c \\
&\leqslant \mathbb{E}[S] \times \log a
\end{aligned}$$

where the last line follows by $c \leqslant 1$ implying $\log c \leqslant 0$. Dividing through by $\log a$ yields the result. We achieve this lower bound only if $q_i = p_i$, that is if $p_i = a^{-s_i}$ for some integers s_i for all p.

In fact, this lower bound must hold for *all* decipherable codes, by McMillan / Karush (1.12).

For the upper bound, we take $s_i = \lceil -\log_a p_i \rceil$. We have $s_i < -\log_a p_i + 1 \implies a^{-s_i} \leqslant p_i$. This means we satisfy Kraft's inequality (1.11), since

$$\sum_{i=1}^{m} a^{-s_i} \leqslant \sum_{i=1}^{m} p_i = 1$$

and therefore there is a prefix-free code with word lengths $s_1 \dots s_m$. Also,

$$\mathbb{E}[S] = \sum_{i=1}^{m} p_i s_i < \sum_{i=1}^{m} p_i(-\log_a p_i + 1) = \frac{H(X)}{\log a} + 1,$$

so our upper bound holds. □

Example 1.20 (Shannon-Fano Coding)

This is an example of a code which follows from the above proof. We set $s_i = \lceil -\log_a p_i \rceil$ and construct a prefix-free code with word lengths $s_1 \dots s_m$ sorted in ascending order, ensuring that previous codewords are not used as prefixes.

Suppose our source emits words $\mu_1 \dots \mu_5$ with probabilities 0.4, 0.2, 0.2, 0.1, and 0.1. Then we construct the binary Shannon-Fano code (with $a = 2$) by taking $s_i = \lceil -\log_2 p_i \rceil$, which is equal to 2, 3, 3, 4, and 4 respectively.

We then have a lot of freedom. At each stage, we may choose *anything* which does not contain a previous word as a prefix. For example, set $\mu_1 \mapsto \texttt{00}$. Then we may choose $\mu_2 \mapsto$ anything of length 3 which does not begin `00`, say `010`. Similarly, $\mu_3 \mapsto \texttt{100}$, then $\mu_4 \mapsto \texttt{1100}$ and lastly $\mu_5 \mapsto \texttt{1110}$, which is a prefix-free and thus decipherable code.

The expected word length $\mathbb{E}[S] = 2.8$. For comparison, the entropy $H(X) \approx 2.122$.

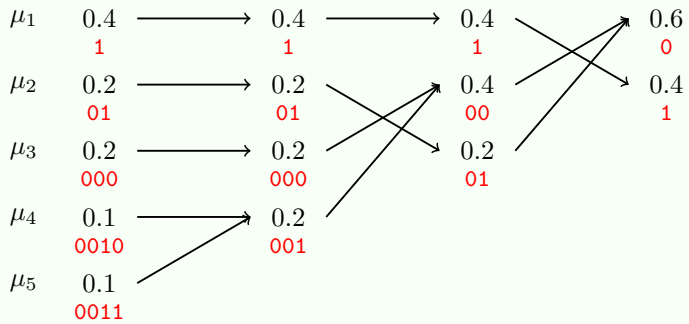
Note: The Shannon-Fano code is not always optimal. However, the next one is!

Example 1.21 (Huffman Coding)

We define Huffman Coding inductively. For simplicity, we take the binary case $a = 2$ again. Order the p_i in descending order $p_1 > \dots > p_m$. Now:

1. If $m = 2$, then assign $s_1 = \texttt{0}$ and $s_2 = \texttt{1}$.
2. If $m > 2$, then find the Huffman code f' for $m - 1$ words: $\mu_1 \dots \mu_{m-2}$ emitted with probabilities $p_1 \dots p_{m-2}$, and a new word ν with probability $p_{m-1} + p_m$. Then, assign words $\mu_1 \dots \mu_{m-2}$ the same codes, and set $f(\mu_{m-1}) = f'(\nu)\texttt{0}$ and $f(\mu_m) = f'(\nu)\texttt{1}$.

This gives a prefix-free code. When some of the p_i are equal, then we can choose how to order them, so Huffman codes are not necessarily unique. In our previous example, this gives:



Now, $\mathbb{E}[S] = 2.2$: notably better than the Shannon-Fano code and remarkably close to $H(X)$.

Let us prove an auxiliary lemma first to show the optimality of the Huffman scheme.

Proposition 1.22 (Sorting and Almost-Equality)

Suppose that $\mu_1 \dots \mu_m$ are emitted with probabilities $p_1 \dots p_m$. Let f be an optimal prefix-free code with word lengths $s_1 \dots s_m$. Then

1. If $p_i > p_j$, then $s_i \leqslant s_j$.
2. There are two codewords of maximal length which are equal up to the last letter.

Proof: (1) is obvious: simply swap the codewords $f(\mu_i)$ and $f(\mu_j)$. This strictly descreases the expected word length, contradicting the optimality of f.

(2) is less obvious. Suppose it is false. Then either there is only one codeword of maximal length, or any two codewords of maximal length differ before the last digit. In either case, remove the last letter of each codeword of maximal length. This maintains the prefix-free condition, and shortens the expected word length, again contradicting the optimality of f. □

Note: Shannon-Fano satisfies the first of these properties, but not necessarily the second. In our example, $f(\mu_4) = \texttt{1100}$ and $f(\mu_5) = \texttt{1110}$ did not satisfy this condition.

Now, we may prove our target theorem.

Theorem 1.23 (Huffman Coding Optimal)

The Huffman coding scheme is optimal (1.18): for words $\mu_1 \dots \mu_m$ emitted with probabilities $p_1 \dots p_m$, it minimises the expected word length $\mathbb{E}[S]$.

Proof: We prove this for the binary $a = 2$ case by showing via induction on m that any Huffman code of size m is optimal. The $m = 2$ case is obvious: the codewords `0` and `1` are minimal.

Suppose $m > 2$. The inductive source emits $\mu_1 \dots \mu_{m-2}$ with probabilities $p_1 \dots p_{m-2}$, and a new word ν with probability $p_{m-1} + p_m$. The Huffman code f_{m-1} is optimal for this source. Now, we construct the Huffman code f_m of size m by extending f_{m-1}. The expected word length satisfies:

$$\mathbb{E}[S_m] = \mathbb{E}[S_{m+1}] + p_{m-1} + p_m$$

Let f'_m be an optimal code for X_m which is prefix-free. Proposition 1.22 then yields that the codewords associated with μ_{m-1} and μ_m are of maximal length and differ only in the last letter. Say these are $y\texttt{0}$ and $y\texttt{1}$ for some string $y \in \Sigma_2^*$. We define a code f'_{m-1} for X_{m-1} with

$$\begin{aligned} f'_{m-1}(\mu_i) &= f'_m(\mu_i) : 1 \leqslant i \leqslant m-2, \\ f'_{m-1}(\nu) &= y. \end{aligned}$$

Then f'_{m-1} is a prefix-free code and the expected-word length satisfies

$$\mathbb{E}[S'_m] = \mathbb{E}[S'_{m-1}] + p_{m-1} + p_m$$

By the induction hypothesis, f_{m-1} is optimal, so $\mathbb{E}[S_{m-1}] \leqslant \mathbb{E}[S'_{m-1}]$. Putting this all together, we obtain $\mathbb{E}[S_m] \leqslant \mathbb{E}[S'_m]$: that is, f_m has word length less than or equal to that of an optimal code for the source X_m. Therefore f_m must itself be optimal, as required. □

Note: Not all optimal codes are Huffman, but (from the proof of the above) it can be seen that any optimal sequence of word lengths $s_1 \dots s_m$ associated with $p_1 \dots p_m$, there is a Huffman code which results in these word lengths.

1.6 Coding Sequences

In Shannon's Noiseless Coding Theorem (1.19) we don't always attain the lower bound $H(X)/\log a$. However, by coding longer *sequences* we can make our code more efficient and closer to this bound.

Example 1.24 (Motivation for Coding Sequences)

Suppose we have a memoryless (1.17) source which emits μ_1 with probability $3/4$ and μ_2 otherwise, the optimal binary code assigns $\mu_1 \mapsto \mathtt{0}$ and $\mu_2 \mapsto \mathtt{1}$.

Consider strings of length 2. We have $\mathbb{E}[S^2] = 2$, since every two-letter input sequence codes to precisely two output letters. Can we beat this? Yes, if we code strings of length 2 directly.

Consider the 4 "letters" $\mu_1\mu_1$, $\mu_1\mu_2$, $\mu_2\mu_1$, and $\mu_2\mu_2$. These have probabilities $9/16$, $3/16$, $3/16$, and $1/16$ respectively. If we apply the Huffman algorithm, we obtain word lengths of 1, 2, 3, and 3, which maps to an expected word length of $\mathbb{E}[S^2] = 27/16 < 2$.

This saving came from mapping $\mu_1\mu_1$, which is a very common sequence, to just one bit. This would not have been possible without word combination! The idea is thus to split our sequences into high probability typical sequences and low probability atypical sequences.

If a coin with $\mathbb{P}[\text{Heads}] = p$ is tossed N times, we expect approximately Np heads and $(1-p)N$ tails. A particular sequence of precisely this many heads and tails has probability:

$$p^{pN}(1-p)^{(1-p)N} = 2^{N(p\log p + (1-p)\log(1-p))} = 2^{-NH(X)}.$$

where X is the result of an individual coin toss. So with high probability, we will get a typical sequence, and its probability will be close to $2^{-NH(X)}$. Can we formalise this idea?

Definition 1.25 (Asymptotic Equipartition Property)

A source $X_1, X_2, X_3 \ldots$ satisfies the *Asymptotic Equipartition Property* (AEP) with constant $H \geqslant 0$ if for all $\varepsilon > 0$ we have that $\exists\, N$ s.t. $(\forall\, n > N,\ \exists\, T_n \subseteq \Sigma^n)$ with:

1. $\mathbb{P}[(X_1 \ldots X_n) \in T_n] > 1 - \varepsilon$
2. $2^{-n(H+\varepsilon)} \leqslant p(x_1, \ldots, x_n) \leqslant 2^{-n(H-\varepsilon)}$ for all $(x_1, \ldots, x_n) \in T_n$

This T_n is called a *typical set*. We then encode the high probability typical sequences carefully and encode the low probability atypical sequences arbitrarily.

Remark 1.26 (AEP Helpful)

For any given $\varepsilon > 0$ and sufficiently large n, we have $p(x_1, \ldots, x_n) \geqslant 2^{-n(H+\varepsilon)}$ for all $\mathbf{x} \in T_n$. Summing over these $\mathbf{x}$, we obtain:

$$1 \geqslant \mathbb{P}[(X_1 \ldots X_n) \in T_n] \geqslant 2^{-n(H+\varepsilon)}\,|T_n| \implies |T_n| \leqslant 2^{n(H+\epsilon)}$$

We encode each of these into string of length r, so we require $a^r > |T_n|$. For atypical sequences, we encode by prefixing a string of length r (not already used) with a string of length n. Then

$$\mathbb{E}[S_n] \leqslant \frac{\lceil n(H+\varepsilon)\rceil}{\log a} + \delta n$$

This is close to the Shannon bound (1.19)! We get $\mathbb{E}[S^n]/n \leqslant H/\log a + \delta'$, where we can make the δ' small, yielding a compact encoding of n-strings.

Now, we consider a property of sources, related to our intuition about the AEP.

Definition 1.27 (Reliable Encodability, Information Rate)

A source $X_1, X_2, \dots$ is *reliably encodable* at rate r if for each n there is $A_n \subseteq \Sigma^n$ with:

1. $\log |A_n| \times (1/n) \to r$ as $n \to \infty$.
2. $\mathbb{P}[(X_1, \dots, X_n) \in A_n] \to 1$ as $n \to \infty$

The *information rate* H of a source is the infimum of all rates at which it is reliably encodable. Then nH is roughly the number of bits required to encode $(X_1, \dots, X_n)$.

Theorem 1.28 (Shannon's First Coding Theorem, 1948)

If a source satisfies the asymptotic equipartition property (1.25) with constant H, then the source has information rate (1.27) equal to H.

Proof: Omitted. □

We now present an alternative definition of the asymptotic equipartition property (1.25).

Definition 1.29 (Asymptotic Equipartition Property)

A source $X_1, X_2, \dots$ satisfies the AEP if for some $H \geqslant 0$, we have

$$-\frac{1}{n} \log p(x_1, \dots, x_n) \xrightarrow{\mathbb{P}} H \quad \text{as } n \to \infty$$

where the arrow refers to *convergence in probability*. This allows the source to take very different values for large n, but only on a set of small probability.

Remark 1.30 (Weak Law of Large Numbers)

Recall that the *weak law of large numbers* states that for any independently and identically distributed sequence of random variables $X_1, X_2, \dots$ with finite expected value $\mathbb{E}[X_i] = \mu$:

$$\frac{1}{n} \sum_{i=1}^{n} X_i \xrightarrow{\mathbb{P}} \mu \quad \text{as } n \to \infty.$$

We can apply this to our toy model of a memoryless (1.17) source, since $p(X_1)$ are iid. random variables, and $p(x_1, \dots, x_n) = p(x_1) \times \cdots \times p(x_n)$, yielding:

$$-\frac{1}{n} \log p(x_1, \dots, x_n) = -\frac{1}{n} \sum_{i=1}^{n} \log p(x_i) \xrightarrow{\mathbb{P}} \mathbb{E}[-\log p(X)] = H(X) \quad \text{as } n \to \infty.$$

Thus any memoryless source satisfies the AEP with constant $H = H(X)$.

Corollary: A memoryless source has information rate equal to its entropy $H(X)$.

2 Error Control Codes

2.1 Binary Codes

Recall our initial schematic of a code from 1.1. In the previous section, we considered the problem of sending coded messages when the channel was *noiseless*, that is when we had a perfect guarantee that the message we sent would be accurately received. Now, we consider the case when this does not hold, because our channel is *noisy*. Examples of such channels can be found in 1.5.

Definition 2.1 (Binary Code)

An $[n, m]$ *binary code* is a subset $C \subseteq \{0,1\}^n$ of size m. We say that C has length n. The elements of C are called *codewords*.

Note: By this definition, since all elements of C have length n, C is a *block code*, where all m of the codewords are of equal length. As seen previously, all block codes are prefix-free.

We use an $[n, m]$-code to send one of m possible messages through a binary symmetric channel (1.5) making n uses of the channel.

Definition 2.2 (Information Rate)

The *information rate* of an $[n, m]$ binary code C is defined as $\rho(C) = \log(m)/n$.

Corollary: Since $C \subseteq \{0,1\}^n$ is of size m, $\rho(C) \leqslant 1$, with equality if and only if $C = \{0,1\}^n$ (or equivalently if $m = 2^n$). Similarly, a code of size $m = 1$ has information rate 0.

The error rate depends on the *decoding rule*. We consider three possible rules:

1. The *ideal observer* decoding rule decodes $x \in \{0,1\}^n$ as the codeword $c \in C$ which maximises the probability $\mathbb{P}[c \text{ sent} \mid x \text{ received}]$.
2. The *maximum likelihood* decoding rule decodes $x \in \{0,1\}^n$ as the codeword $c \in C$ which maximises the probability $\mathbb{P}[x \text{ received} \mid c \text{ sent}]$.
3. The *minimum distance* decoding rule decodes $x \in \{0,1\}^n$ as the codeword $c \in C$ which has the fewest digits changed: that is, minimising $\#\{1 \leqslant i \leqslant n : x_i \neq c_i\}$.

Note: For each of these, some convention is needed in case of a tie (when the codeword chosen is not unique). We could choose one at random, or arbitrarily yet consistently, or ask for the message to be sent again.

Proposition 2.3 (Decoder Agreement 1)

If all messages in C are equally likely to be sent, then the *ideal observer* decoder method and the *maximum likelihood* decoder method agree on how to decode any received message.

Proof: By Bayes' rule, we can calculate the probability:

$$\mathbb{P}[c \text{ sent} \mid x \text{ received}] = \frac{\mathbb{P}[c \text{ sent}, x \text{ received}]}{\mathbb{P}[x \text{ received}]} = \frac{\mathbb{P}[c \text{ sent}]}{\mathbb{P}[x \text{ received}]} \times \mathbb{P}[x \text{ received} \mid c \text{ sent}]$$

but having received any x, this last fraction is equal for all c, and so the two probabilities must be equal to each other. The methods thus assign equal "scores" to all codewords, so must agree. □

Now, we use the *minimum distance* rule as the basis for a definition.

Definition 2.4 (Hamming Distance)

For $x, y \in \{0,1\}^n$, the *Hamming distance* between x and y is the scoring rule used by the minimum distance observer $d(x,y) = \#\{1 \leqslant i \leqslant n : x_i \neq c_i\}$. Notice that this is a metric!

Proposition 2.5 (Decoder Agreement 2)

If $p < 1/2$, then the *maximum likelihood* decoder method and the *minimum distance* decoder method agree on how to decode any received message.

Proof: Suppose $d(x,c) = r$. Then we can calculate the probability explicitly as:

$$\mathbb{P}[x \text{ received} \mid c \text{ sent}] = p^r(1-p)^{r-n} = (1-p)^n \times \left(\frac{p}{1-p}\right)^{n-r}$$

When $p < 1/2$, this last fraction is less than 1. Therefore choosing c to maximise this probability is the same as choosing c to minimise $d(x,c)$. □

Note: As mentioned in 1.5, we usually take $p < 1/2$ in general. If $p > 1/2$, then our bit is flipped most of the time, so it would make more sense to send the opposite bit, in which case our bit is now mostly correct (as if $p < 1/2$). If $p = 1/2$, we have an entirely useless channel which simply outputs a stream of random bits with no correlation to what we sent, which is uninteresting.

Example 2.6 (Encoding Codewords)

Suppose we have the codewords "000" and "111" which are sent with probabilities $\alpha = 0.9$ and $1-\alpha = 0.1$ respectively. We use a BSC with error probability $p = 1/4$.

If we receive the string "110", how should we decode it?

Clearly, the *minimum distance* decoder (and therefore the *maximum likelihood* decoder too, by Proposition 2.5) will decode the string as "111".

However, the *ideal observer* decoder will calculate the odds ratio as:

$$\underbrace{9:1}_{\text{prior of 000 vs 111}} \times \underbrace{(3/64):(9/64)}_{\text{odds of two flips vs one flip}} = \underbrace{3:1}_{\text{posterior odds ratio}}$$

giving a probability of $3/4$ that "000" was sent, and thus choosing it as the most likely of the two codewords to have been sent, having received "110".

Note: The *ideal observer* rule is also known as the *minimum error* rule. It seems much better, but it requires knowing the prior probabilities of each codeword being sent. From now on, we use the other two methods, since they are equivalent.

Definition 2.7 (Error Detecting/Correcting)

C is d-error *detecting* if changing at most d letters of a codeword never produces a different codeword. Equivalently, this is the minimum separation distance of C.

C is e-error *correcting* if the knowledge that the string received as at most e errors is sufficient to determine with certainty which codeword was sent.

We often consider the *repetition code* of length n. Here, the codewords we want to send are just the n-long strings of all 0s and all 1s, where we simply repeat a single bit we want to send. This is an $[n, 2]$ binary code. We can detect $n-1$ errors, and correct anything less than $n/2$ errors, which is fairly good for a code! Unfortunately, the information rate (2.2) is only $1/n$.

Note: The information rate seems like a good definition! In this example we used n bits of channel space to send 1 bit of actual information, and had an information rate of $1/n$. In fact, this holds in general: the information rate can be thought of as the "bits per bit" of a code.

Example 2.8 (Simple Parity Check Code)

The *simple parity check code* of length n, also known as the *paper tape code*, is another common code example. Here, we view the first $n-1$ bits as the actual information to communicate, and use the last bit as a free bit to enforce the rule that the total number of `1`s in the codeword is even. That is:

$$C = \left\{ (x_1 \dots x_n) \in \{\texttt{0}, \texttt{1}\}^n : \sum_{i=1}^{n} x_i \equiv 0 \pmod 2 \right\}$$

is the set of codewords, which is an $[n, 2^{n-1}]$ code. It is 1-error detecting, but it cannot correct any errors (0-error correcting). Its information rate is $1 - 1/n$, which is a lot better.

Note: Suppose we change our code C to use the same permutation to reorder each codeword. Then we get a code with the same information rate, error detection capabilities, and so forth. We say such a code is *permutationally equivalent*.

In the 1940s, Richard Hamming was working at Bell Labs on an old computer which used punch cards to store and run code. Since users of punch cards were prone to making errors, there were safety checks built in to the machines, so that they could detect malformed input, and loudly alert the operators with bright flashing lights and loud noises. Hamming was frustrated by this, and was said to have remarked "Damn it, if the machine can detect the error, why can't it correct it?"

This experience influenced him to create the original *error-correcting Hamming code*.

Example 2.9 (Hamming's Original 1950 Code)

Let $C \in \{\texttt{0}, \texttt{1}\}^7$ be defined by the 7-tuples which satisfy the congruences:

$$\begin{aligned} c_1 + c_3 + c_5 + c_7 &\equiv 0 \pmod 2 \\ c_2 + c_3 + c_6 + c_7 &\equiv 0 \pmod 2 \\ c_4 + c_5 + c_6 + c_7 &\equiv 0 \pmod 2 \end{aligned}$$

Since there are three of these congruences, the size of C is $2^{7-3} = 16$. This means that C is a [7, 16] code, and thus has information rate $4/7$.

Suppose we receive some $x \in \{\texttt{0}, \texttt{1}\}^7$. Then we form the *syndrome* $z_x = (z_1, z_2, z_4)$, where:

$$\begin{aligned} z_1 &= x_1 + x_3 + x_5 + x_7 \\ z_2 &= x_2 + x_3 + x_6 + x_7 \\ z_4 &= x_4 + x_5 + x_6 + x_7 \end{aligned}$$

with addition taken modulo 7. For any $x \in C$, by construction we have $z_x = (0, 0, 0)$.

If $d(x, c) = 1$ for some $c \in C$, then the place where they differ is given by $z_1 + 2z_2 + 4z_3$. This is because if $x = c + e_i$, where e_i is a vector with all `0`s except for a `1` in the i^{th} place, then the syndrome of x is the syndrome of e_i, which is the binary expansion of i for all $1 \leqslant i \leqslant 7$.

This is because, for example, x_3 appears in the definitions of z_1 and z_2 only, since $3 = \texttt{110}_2$ and so only bits 1 and 2 would be affected.

Thus our code C corrects any single error! However, it doesn't correct two: for example, the string `1110000` $\in C$ could be corrupted to `1000000`, which would be decoded as `0000000`.

Now recall that in the definition of the Hamming distance (2.4), we stated that this was a metric. We now prove this formally.

Proposition 2.10 (Hamming Metric)

The *Hamming distance* $d(x, y) = \#\{1 \leqslant i \leqslant n : x_i \neq c_i\}$ is a metric on $\{0, 1\}^n$.

Proof: Clearly $d(x, y) \geqslant 0$, as the count of a set. If $d(x, y) = 0$, then x and y differ in zero places, and so they must be the same: conversely, $d(x, x)$ is clearly 0. Also, the symmetry of the definition gives us the relation $d(x, y) = d(y, x)$. So we only need to show the triangle inequality, using:

$$\{1 \leqslant i \leqslant n : x_i \neq z_i\} \subseteq \{1 \leqslant i \leqslant n : x_i \neq y_i\} \cup \{1 \leqslant i \leqslant n : y_i \neq z_i\}$$

which yields $d(x, z) \leqslant d(x, y) + d(y, z)$. Equivalently, it is the sum metric on n copies of the discrete metric on $\{0, 1\}$, which is therefore a metric itself. □

Definition 2.11 (Minimum Distance)

The *minimum distance* of a code $C \subseteq \{0, 1\}^n$ is the smallest Hamming distance between two distinct codewords. An $[n, m]$ code with minimum distance d is sometimes referred to as an $[n, m, d]$ code. For example, Hamming's original code is a [7, 16, 3] code.

Note: We have $m \leqslant 2^n$, with equality if and only if $C = \{0, 1\}^n$. This is called the *trivial code*.

From this definition, we can prove some bounds on how "good" a code can be (2.7).

Proposition 2.12 (Error Bounds)

Let C be a code with minimum distance d. Then:

(i) C can always detect up to $d - 1$ errors, but not necessarily d errors.

(ii) C can always correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors, but not necessarily more.

Proof: Suppose $x \in \{0, 1\}^n$ and $c \in C$ with $1 \leqslant d(x, c) \leqslant d - 1$. Then $x \notin C$, as otherwise the minimum distance would not be d. Therefore C can detect up to $d - 1$ errors. But if $c_1, c_2 \in C$ with $d(c_1, c_2) = d$, then c_1 can be corrupted to c_2 with just d errors, which the code would not be able to detect. So d errors cannot always be detected.

Now, let $e = \left\lfloor \frac{d-1}{2} \right\rfloor$, so $e \leqslant \frac{d-1}{2} \leqslant e + 1$, or equivalently $2e < d \leqslant e + 1$.

Then take $x \in \{0, 1\}^n$. If there is some $c_1 \in C$ with $d(x, c_1) \leqslant e$, we want to show $d(x, c_2) > e$ for all $c_2 \neq c_1$ in C. This is given directly by the triangle inequality:

$$d(x, c_2) \geqslant d(c_1, c_2) - d(x, c_1) \geqslant d - e > e.$$

Thus C is e-error correcting. However, take $c_1, c_2 \in C$ with $d(c_1, c_2) = d$. Let x differ from c_1 in precisely $e + 1$ places where c_1 and c_2 also differ. Then $d(x, c_1) = e + 1$, and we have

$$d(x, c_2) = d - (e + 1) \leqslant e + 1$$

so both c_1 and c_2 can be corrupted to x with $e + 1$ errors. Therefore C cannot correct $e + 1$ errors, and so our bounds are tight. □

Corollary: The *repetition code* is an $[n, 2, n]$ code, so detects $n - 1$ errors and corrects $\left\lfloor \frac{n-1}{2} \right\rfloor$.

Corollary: The *paper tape code* is an $[n, 2^{n-1}, 2]$ code, so detects one error but corrects none.

Corollary: The *original Hamming code* is a [7, 16, 3] code, as mentioned earlier.

2.2 New Codes from Old

Given an $[n, m, d]$ code, we might want to transform it: making the code "safer" by enabling it to detect or correct more errors, but at the cost of increasing the length of the codewords. Conversely, we may want to go the other way, sacrificing robustness for efficiency.

Definition 2.13 (Parity Extension, Punctured Code, Shortened Code)

Let C be an $[n, m, d]$ code. Then The *parity extension* of C is

$$\bar{C} = \left\{ (c_1, c_2, \dots c_n, \sum_{i=1}^{n} c_i) : (c_1 \dots c_n) \in C \right\},$$

where the addition is taken modulo 2. That is, the new code is the old code, where each of the codewords has an extra bit added as a parity check. This makes $\bar{C}$ an $[n+1, m, d']$ code, where $d \leqslant d' \leqslant d+1$, depending on the parity of d.

This code is longer but potentially more error-detecting. Conversely, the *punctured code* goes the other direction, and deletes the i^{th} letter from each codeword. This forms a code which is one bit shorter, but possibly combines two codewords, unless no two codewords differ only in this letter. A sufficient condition to ensure that this does not happen is to enforce $d \geqslant 2$.

Similarly, we define the *shortened code* for a fixed $a \in \{0, 1\}$ and $1 \leqslant i \leqslant n$. We take all the codewords in C, and remove the i^{th} letter, given that it is an a. For some choice of a, this will retain at least $\lceil m/2 \rceil$ codewords from the original C.