Notes on Information Theory and Coding.

Henk Bruin

December 15, 2020

1 Automata

In this section we discuss Turing machines and variations of it, and ask the question what languages they can recognize or generate. The terminology is not entirely consistent in the literature, so some of the below notions may be called differently depending on which book you read.

1.1 Turing Machines

A Turing machine is a formal description of a simple type of computer, named after the British mathematician Alan Turing (1912-1954). He used this in theoretic papers to explore the limits what is computable by computers and what is not. For us, the size of a Turing machine that can recognize words in a language $\mathcal{L}(X)$, or reject words that don't belong to $\mathcal{L}(X)$, is a measure for how complicated a subshift is. In fact, a subshift is called **regularly enumerable** in the Chomsky hierarchy if its language can be recognized by a Turing machine.

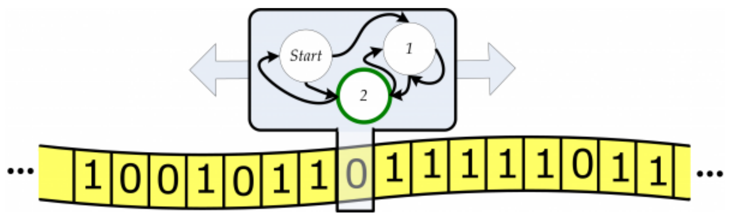


Figure 1: Alan Turing (1912-1954) and his machine.

A Turing machine has the following components:

- A tape on which the input is written as a word in the alphabet $\{0, 1\}$.
- A reading device, that can read a symbol at one position on the tape at the time. It can also erase the symbol and write a new one, and it can move to the next or previous position on the tape.

- A finite collection of states $S_1, \ldots, S_N$, so N is the size of the Turing machine. Each state comes with a short list of instructions:
  - read the symbol;
  - replace the symbol or not;
  - move to the left or right position;
  - move to another (or the same) state.

  One state, say S_1, is the **initial state**. One (or several) states are **halting states**. When one of these is reached, the machine stops.

Example 1.1. *The following Turing machine rejects tape inputs that do not belong to the language of the Fibonacci shift. Let s be the symbol read at the current position of the tape, starting at the first position. We describe the states:*

S_1: *If $s = 0$, move to the right and go to State S_1. If $s = 1$, move to the right and go to State S_2.*

S_2: *If $s = 0$, move to the right and go to State S_1. If $s = 1$, go to State S_3.*

S_3: *Halt. The word is rejected, see Figure 2.*

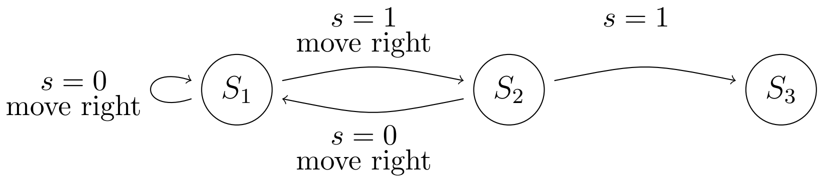


Figure 2: A Turing machine accepting words from the Fibonacci shift.

Rather than an actual computing devise, Turing machines are a theoretical tool to study which types of problems are in principle solvable by a computer. For instance, we call a sequence $(x_n)_{n \geq 0}$ automatic if there is a Turing machine that produces x_n hen the binary expansion of n is written on the input tape. A number is automatic if its digits form an automatic sequence. Given that there are only countably many Turing machines, there are only countably many automatic numbers. Most reals are not automatic: no algorithm exists that can produce all its digits.

Another question to ask about Turing machines, or any computer program in general, is whether it will stop in finite time for a given input. This is the **halting problem**. Using an advanced version of the Russell paradox, Turing proved in 1936 [28] that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist. However, in theory it is possible to create a **universal Turing machine** that does the work of all Turing machines together. Quoting Turing [28, page 241-242]:

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with the tape on the beginning of which is written the string of quintuples separated by semicolons of some computing machine $\mathcal{M}$, then $\mathcal{U}$ will compute the same sequence as $\mathcal{M}$.

For this $\mathcal{U}$ doesn't need to combine the programs of the countably many separate Turing machine. Instead, the separate Turing machines $\mathcal{M}$ are to be encoded on the tape in some standardized way, and they will be read by $\mathcal{U}$ together with the input of for $\mathcal{U}$. The program of $\mathcal{U}$ then interpret $\mathcal{M}$ and executes it. Universal Turing machines didn't stay theoretical altogether. Shannon posed the question how small universal Turing machines can be. Marvin Minsky [19] constructed a version on a four-letter alphabet with seven states. Later construction emerged from cellular automata. This was improved to a 3-letter alphabet by Alex Smith in 2007, but his prove is in dispute. A smaller alphabet is impossible [20][1].

Exercise 1.2. *Suppose two positive integers m and n are coded on a tape by first putting m ones, then a zero, then n ones, and then infinitely many zeros. Design Turing machines that compute $m+n$, $|m-n|$ and mn so that the outcome is a tape with a single block of ones of that particular length, and zeros otherwise.*

1.2 Finite Automata

A finite automaton (FA) is a simplified type of Turing machine that can only read a tape from left to right, and not write on it. The components are

$$M = \{Q, \mathcal{A}, q_0, q, f\} \tag{1}$$

where

Q	$=$	collection of **states** the machine can be in.
$\mathcal{A}$	$=$	the alphabet in which the tape is written.
q_0	$=$	the initial state in Q.
H	$=$	collection of halting states in Q; the FA halts when it reaches one.
f	$=$	is the rule how to go from one state to the next when reading a symbol $a \in \mathcal{A}$ on the tape. Formally it is a function $Q \times \mathcal{A} \to Q$.

A language is **regular** if it can be recognized by a finite automaton.

Example 1.3. *The even shift is recognized by the following finite automaton with $Q = \{q_0, q_1, q_2, q_3\}$ with initial state q_0 and final states q_2 (rejection) and q_3 (acceptance). The tape is written in the alphabet $\mathcal{A} = \{0, 1, b\}$ where b stands for a* blank *at the end of the input word. The arrow $q_i \to q_j$ labeled $a \in \mathcal{A}$ represents $f(q_i, a) = q_j$.*

[1]This proof by Morgenstern is based on an unpublished result by L. Pavlotskaya from 1973.

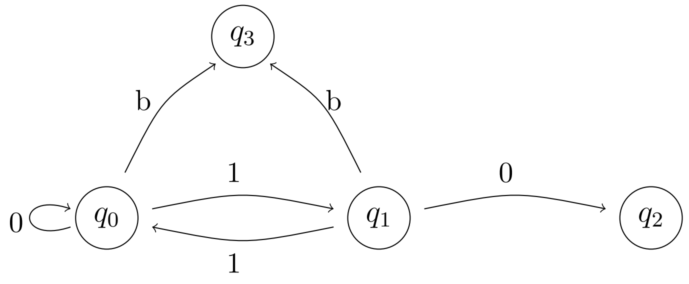


Figure 3: Transition graph for a finite automaton recognizing the even shift.

This example demonstrates how to assign a edged-labeled transition graph to a finite automaton, and it is clear from this that the regular languages are precisely the sofic languages.

It is frequently easier, for proofs or constructing compact examples, to allow finite automata with multiple outgoing arrows with the same label. So, if we are in state q, read symbol a on the input tape, and there is more than one outgoing arrow with label a, then we need to make choice. For computers, making choices is somewhat problematic - we don't want to go into the theoretical subtleties of random number generators - but if you take the viewpoint of probability theory, you can simply assign equal probability to every valid choice, and independent of the choices you may have to make elsewhere in the process. The underlying stochastic process is then a discrete Markov process.

Automata of this type are called **non-deterministic finite automata** (NFA), as opposed to **deterministic finite automata** (DFA), where never a choice needs to be made. A word is accepted by an NFA if there is a positive probability that choices are made that parse the word until the end without halting or reaching a rejecting state.

We mention without proof (see [13, page 22] or [2, Chapter 4]):

Theorem 1.4. *Let $\mathcal{L}$ be a language that is accepted by a non-deterministic finite automaton. Then there is a deterministic finite automaton that accepts $\mathcal{L}$ as well.*

Corollary 1.5. *Let $w^R = w_n \dots w_1$ stand for the* **reverse** *of a word $w = w_1 \dots w_n$. If a language $\mathcal{L}$ is recognized by a finite automaton, then so is its reverse $\mathcal{L}^R = \{w^R : w \in \mathcal{L}\}$.*

Proof. Let $(\mathcal{G}, \mathcal{A})$ the edge-labeled directed graph representing the FA for $\mathcal{L}$. Reverse all the arrows. Clearly the reverse graph $(\mathcal{G}^R, \mathcal{A})$ in which the directions of all arrows are reversed and the final states become initial states and vice versa, recognizes $\mathcal{L}^R$. However, even if in $\mathcal{G}$, every outgoing arrow has a different label (so the FA is deterministic), this is no longer true for $(\mathcal{G}^R, \mathcal{A})$. But by Theorem 1.4 there is also an DFA that recognizes $\mathcal{L}^R$. □

Sometimes it is easier, again for proofs or constructing compact examples, to allow finite automata to have transitions in the graph without reading the symbol on the

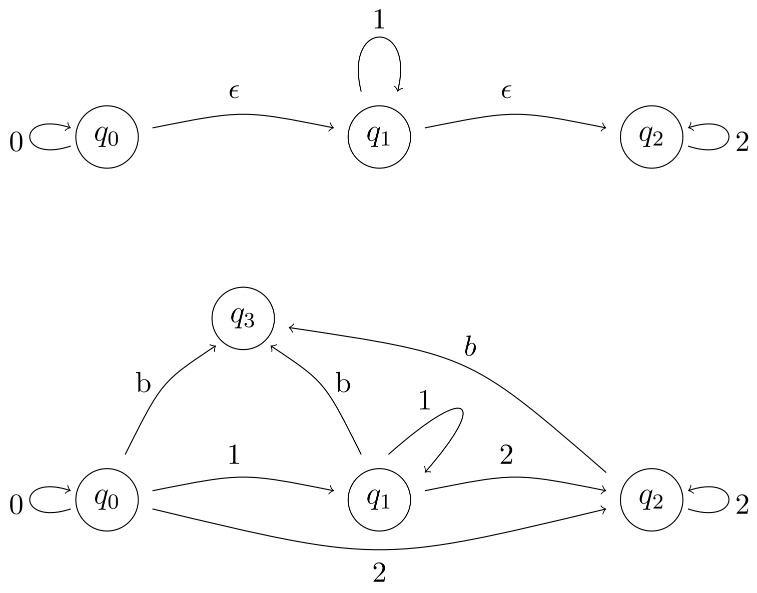


Figure 4: Finite automata recognizing $\mathcal{L} = \{0^k 1^l 2^m : k, l, m \geq 0\}$.

input take (and moving to the next symbol). Such transitions are called **ϵ-moves**. Automata with ϵ-moves are almost always non-deterministic, because if a state q has an outgoing arrow with label a and an outgoing arrow with label ϵ, and the input tape reads a, then still there is the choice to follow that a-arrow or the ϵ-arrow.

Example 1.6. *The follow automata accept the language $\mathcal{L} = \{0^k 1^l 2^m : k, l, m \geq 0\}$, see Figure 4. The first is with ϵ-moves, and it stops when the end of the input is reached (regardless which state it is in). That is , if the FA doesn't halt before the end of the word, then the word is accepted. The second is deterministic, but uses a blank b at the end of the input. In either case q_0 is the initial state.*

Again without proof (see [13, page 22]):

Theorem 1.7. *Let $\mathcal{L}$ be a language that is accepted by a finite automaton with ϵ-moves. Then there is a non-deterministic finite automaton without ϵ-moves that accepts $\mathcal{L}$ as well.*

A **deterministic finite automaton with output** (DFAO) is an septuple

$$M_O = \{Q, \mathcal{A}, q_0, H, f, \tau, \mathcal{B}\}, \tag{2}$$

where the first five components are the same as for an FA in (1), and $\tau : Q \to \mathcal{B}$ is an output function that gives a symbol associated to each state in Q. It can be the writing device for a Turing machine. For a word $w \in \mathcal{A}^*$, we let $\tau(q_0, w) := t(q)$ be the symbol that is read off when the last letter of w is read (or when the automaton halts).

The following central notion was originally due to Büchi [4], see also the monograph by Allouche & Shallit [2].

Definition 1.8. *Let $\mathcal{A} = \{0, 1, \dots, N-1\}$, and let $[n]_{\mathcal{A}}$ denote the integer n expressed in base N. A sequence $x \in \mathcal{B}^{\mathbb{N}}$ is called an N-***automatic sequence** *if $x_n = \tau(q_0, [n]_{\mathcal{A}})$ for all $n \in \mathbb{N}$.*

Example 1.9. *The automaton in Figure 5 assigns the symbol 1 to every word $w \in \mathcal{L}(X_{even})$ of the one-sided even shift and the symbol 0 to words $w \in \{0,1\}^{\mathbb{N}} \setminus \mathcal{L}(X_{even})$. The output function $\tau : Q \to \{0,1\}$ is indicated by the second symbol at each state. The only halting state is q_3.*

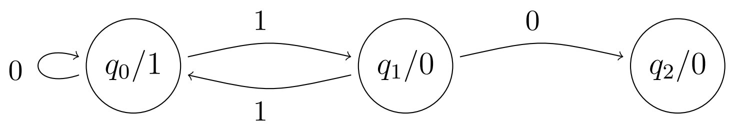


Figure 5: The DFAO for the even shift.

As such, the sequence $x \in \{0,1\}^{\mathbb{N}}$ defined as

$$x_n = \begin{cases} 1 & n \textit{ contains only even blocks of } 1 \textit{ in its binary expansion,} \\ 0 & \textit{otherwise,} \end{cases}$$

is an automatic sequence.

All sequence that are eventually periodic are automatic. In [2, Section 5.1], the Thue-Morse sequence $\rho_{\text{TM}} = 0110\,1001\,10010110\,10\dots$ is used as an example of an automatic sequence, because of its characterization as $x_n = \#\{1\text{s in the binary expansion of } n-1\} \bmod 2$, see Figure 6.

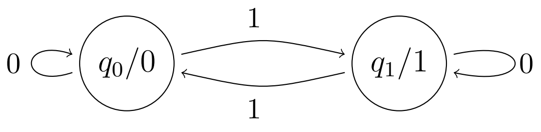


Figure 6: The DFAO for the Thue-Morse sequence.

Sturmian sequences are in general not; only if they are of the form $x_n = \lfloor n\alpha + \beta \rfloor \bmod N$ for some $\alpha \in \mathbb{Q}$, see [1]. Some further results on automatic sequences are due to Cobham. The first is Cobham's Little Theorem see [3, 5].

Theorem 1.10. *The fixed point ρ of a substitution χ on $\mathcal{A} = \{0, \dots, N-1\}$ is an N-automatic sequence, and so is $\psi(\rho)$ for any substitution $\psi : \mathcal{A} \to \mathcal{B}^*$.*

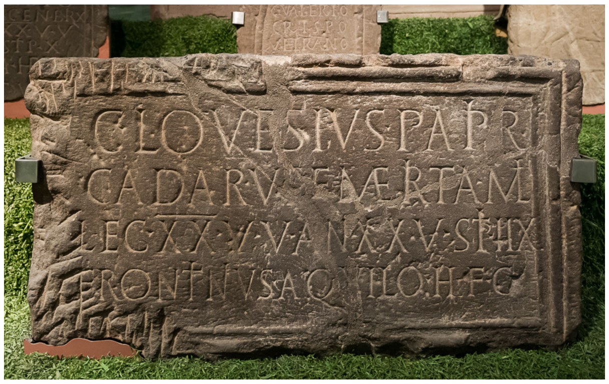
The other is Cobham's Theorem see [5, 15].

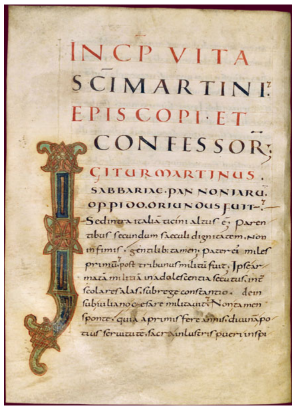
Theorem 1.11. *If $2 \leq M, N \in \mathbb{N}$ are multiplicative independent, i.e., $\frac{\log M}{\log N} \notin \mathbb{Q}$, then the only sequences which are both M-automatic and N-automatic are eventually periodic.*

Since all automatic sequence can basically written as in Theorem 1.10 (cf. [17]), this gives retrospectively information on substitution shifts, see Durand [6, 8, 7]

2 Data Compression

The wish to shorten texts is as old as writing itself. For example, tomb stones were seldom large enough to contain the intended message, so people resorted to (standard) abbreviations, see Figure 7. In the middle ages, vellum was expensive (it still is), so also here a variety of abbreviations was in use.





G(aius) Lovesius Papir(ia) (tribu)
Cadarus Emerita mil(es)
Leg(ionis) XX V(aleriae) V(ictricis)
an(norum) XXV stip(endiorum) IIX
Frontinius Aquilo h(eres)
f(aciendum) c(uravit)

Inc(i)p(it) vita sci (= sancti)
Martini episcopi et confessor(is).
Igitur Martinus Sabbariae
Pannoniaru(m) oppido oriundus fuit,
sed intra Italiam Ticini altus e(st).
parentibus secundum saeculi ...

Figure 7: A Roman tombstone and Carolingian manuscript.

Naturally, writing itself is a means of coding messages, but over the years other methods of coding have been developed for special purposes, such as data compression, storage and transmission by computers (including bar-codes and two-dimensional bar-codes), for secrecy and many other needs.

In 1821, Louis Braille developed the system of now named after him, after an earlier scheme that Charles Barbier devised for military use (Napoleon wanted a method for his soldiers to communicate in silence and without (day)light). Braille, who was a teacher at the Royal Institute for the Blind in Paris, designed his code of raised dots in 2×3-cells that you can feel sliding your finger over the paper. There is a version of Braille, designed by Abraham Nemeth in 1952 (but with later revisions), intended to read mathematical texts. Compared to Braille, some symbols are coded differently (see Figure 8) and for more complicated symbols or expressions multiple six-dot patterns are combined.

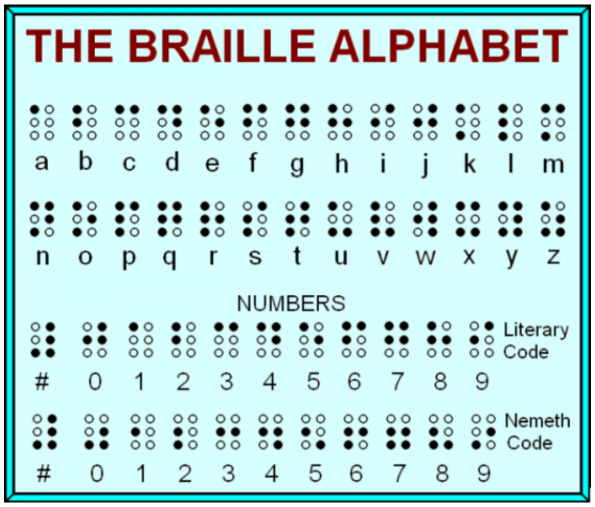

THE BRAILLE ALPHABET
a b c d e f g h i j k l m
n o p q r s t u v w x y z
NUMBERS
# 0 1 2 3 4 5 6 7 8 9
Literary Code
# 0 1 2 3 4 5 6 7 8 9
Nemeth Code

Figure 8: Louis Braille (1809 – 1852) and his code.

When paper eventually became cheap and widely available in the 19th century, data compression of this sort wasn't important anymore, but stenography (for taking speedy dictation or making eye-witness reports to be worked out in full later) was probably the most common form of data compression. With the advent of telegraphy and radio, there as a new reason to compress messages again. In Morse code (see Figure 9), the codes of the most frequently used letters are the shortest.

Morse code was developed in 1836 by Samuel Morse for telegraph[2] (and after its invention also radio) communication. It encodes every letter, digit and some punctuation marks by strings of short and long sounds, with silences between the letters, see Figure 9.

The lengths of the codes were chosen to be inverse proportional to the frequency the letters in the English language, so that the most frequent letter, E and T, get the shortest codes • and •, also to see that for other languages than English, Morse code is not optimal. This is one of the first attempts at data compression. The variable length of codes means, however, that Morse code is not quite a sliding block code.

Mechanization of the world increased the need for coding. The first computers in the middle of the 20th century were used for breaking codes in WWII. A little later, Shannon thought about optimizing codes for data transmission, naturally for

[2]In fact, Morse collaborated with Joseph Henry and Alfred Vail on the development of the telegraph, and they were not the only team. For example, in Göttingen, Gauß and Weber were working on early telegraph and telegraph codes since 1833.

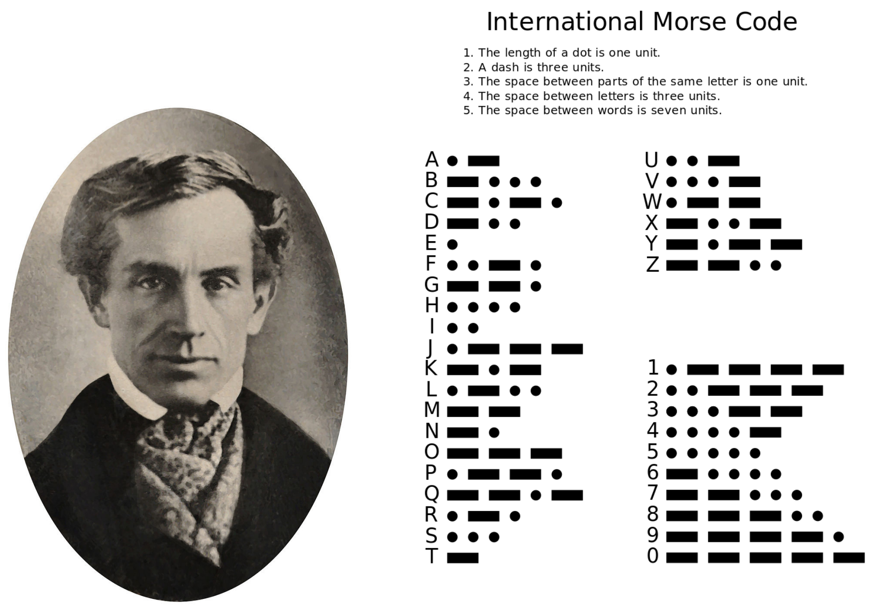


Figure 9: Samuel Morse (1791 – 1872) and his code.

telegraph and telephone usage, see Section 2. But he was visionary enough to imagine that computers would soon become the main purpose of coding. In fact, he would see a large part of his predictions come true because he lived until 2001. The digital representation of text, that is representing letters as strings of zeros and ones, was standardized in the 1960s. ASCII stands for American Standard Code for Information Interchange. Developed in 1963, it assigns a word in $\{0,1\}^7$ to every symbol on the keyboard, including several control codes such as tab, carriage return, most of which are by now obsolete. These seven digit codes allow for $2^7 = 128$ symbols, and this suffices for most purposes of data transmission and storage. Different versions were developed for different countries and makes of computers, but as in Braille, a text is coded in symbol strings, letter by letter, or in other words by a sliding block code of window size 1.

By this time bank transactions started to be done by telephone line, from computer to computer, and cryptography became in integral part of commerce, also in peace-time. Sound and music recordings became digital, with digital radio and CD- and DVD-players, and this cannot go without error correcting codes. For CD's, a cross interleaved Reed-Somolon code is used, to correct errors that are inevitably caused by micro-scratches on the disc. CDs are particularly prone to burst errors, which are longer scratches in the direction of the track, which wipe out longer strings of code. For this reason, the information is put twice on the disc, a few seconds apart (this is what the "interleaved" stands for), which causes the little pause before a CD starts to play.

2.1 Shannon's Source Code Theorem

This "most frequent $\Leftrightarrow$ shortest code" is the basic principle that was developed mathematically in the 1940s. The pioneer of this new area of information theory was Claude Shannon (1916–2001) and his research greatly contributed to the mathematical notion of entropy.

Figure 10: Claude Shannon (1916–2001) and Robert Fano (1917–2016).

In his influential paper [26], Shannon set out the basic principles of information theory and illustrated the notions of entropy and conditional entropy from this point of view. The question is here how to efficiently transmit messages through a channel and more complicated cluster of channels. Signals are here strings of symbols, each with potentially its own transmission time and conditions.

Definition 2.1. *Let $W(t)$ be the allowed number of different signals that can be transmitted in time t. The* **capacity** *of the channel is defined as*

$$Cap = \lim_{t\to\infty} \frac{1}{t} \log W(t). \tag{3}$$

Note that if $X = \mathcal{A}^*$ is the collection of signals, and every symbol takes τ time units to be transmitted, then $W(t) = \#\mathcal{A}^{\lfloor t/\tau \rfloor}$ and $\text{Cap} = \frac{1}{\tau}\log \#\mathcal{A}$. This $W(t)$ doesn't mean the number of signals that can indeed be transmitted together in a time interval of length t, just the total number of signals each of which can be transmitted in a time interval of length t. We see thus that the capacity of a channel is the same as the entropy of the language of signals, but only if each symbol needs the same unit transmission time. If, on the other hand, the possible symbols $s_1, \dots, s_n$ have transmission times $t_1, \dots, t_n$, then

$$W(t) = W(t-t_1) + \cdots + W(t-t_n),$$

where the j-th term on the right hand side indicates the possible transmissions after first transmitting s_j. Using the ansatz $W(t) = ax^t$ for some $x \geq 1$, we get that the leading solution λ of the equation

$$1 = x^{-t_1} + \cdots + x^{-t_n},$$

solves the ansatz, and therefore $\text{Cap} = \log \lambda$. A more general result is the following:

Theorem 2.2. *Suppose the transmission is done by an automaton with d states, and from each state i any signal from a different group $S_{i,j}$ can be transmitted with transmission time $t_{i,j}^s$, after which the automaton reaches state j, see Figure 11. Then the capacity of the channel is $Cap = \log \lambda$ where λ is the leading root of the equation*

$$\det\left(\sum_{s\in S_{i,j}} x^{-t_{i,j}^s} - \delta_{i,j}\right) = 0,$$

where $\delta_{i,j}$ indicates the Kronecker delta.

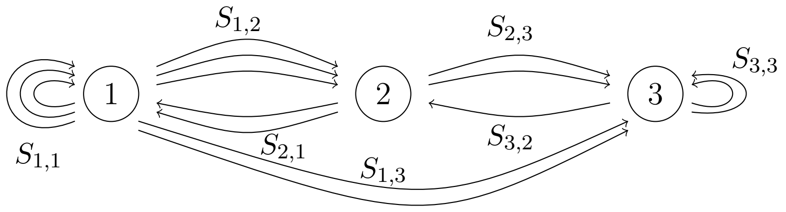


Figure 11: A transmission automaton.

Proof. Let $W_j(t)$ denote the number of possible signals of length t ending at state j, so

$$W_j(t) = \sum_{i,s\in S_{i,j}} W_i(t - t_{i,j}^s). \tag{4}$$

Use the ansatz $W_j(t) = a_j x^t$. Then (4) gives

$$\sum_{i,s\in S_{i,j}} a_i\left(x^{-t_{i,j}^s} - \delta_{i,j}\right) \qquad \text{for all } j \le d.$$

These equations can only have a simultaneous nontrivial solution $(a_1, \dots, a_R)$ if

$$\det\left(\sum_{s\in S_{i,j}} x^{-t_{i,j}^s} - \delta_{i,j}\right) = 0.$$

Therefore Cap $= \lim_{t\to\infty} \frac{1}{t}\sum_{j=1}^d a_j x^t = \log x$. $\square$

It makes sense to expand this idea of transmission automaton to a Markov chain, where each transmission $s \in S_{i,j}$ happens with a certain probability $p_{i,j}^s$ such that $\sum_{j=1}^R \sum_{s\in S_{i,j}} p_{i,j}^s = 1$ for every $1 \le i \le d$. For example, if the states $i \in \mathcal{A}$ are the letters in the English alphabet, the transmissions are single letters $j \in \mathcal{A}$ and the probabilities $p_{i,j}^j$ are the diagram frequencies of ij, conditioned to the first letter i.

Ergodicity is guaranteed if the graph of this automaton is strongly connected. Also, if π_j is the stationary probability of being in state $j \in \{1, \dots, d\}$, then

$$\pi_j = \sum_{i=1}^{d} \pi_i \sum_{s \in S_{i,j}} p^s_{i,j} \qquad \text{for all } j \in \{1, \dots, d\},$$

see the Perron-Frobenius Theorem.

Shannon introduced an **uncertainty function** $H = H(p_1, \dots, p_d)$ as a measure of the amount of uncertainty of the state we are in, if only the probabilities $p_1, \dots, p_d$ of the events leading to this state are known. This function should satisfy the following rules:

1. H is continuous in all of its arguments;
2. If $p_i = \frac{1}{d}$ for all $d \in \mathbb{N}$ and $i \in \{1, \dots, d\}$, then $d \mapsto E(d) := H(\frac{1}{d}, \dots, \frac{1}{d})$ is increasing;
3. If the tree of events leading to the present state is broken up into subtrees, the uncertainty H is the weighted average of the uncertainties of the subtrees (see Figure 12):

$$H(p_1, \dots, p_d) = H(p_1 + p_2, p_3, \dots, p_d) + (p_1 + p_2) H(p, 1 - p).$$

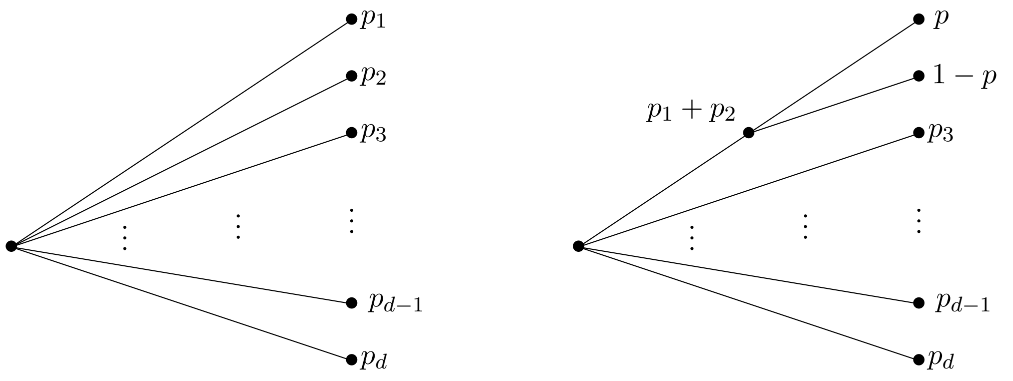


Figure 12: Illustrating rule (3) of the uncertainty function.

Theorem 2.3. *EFor every uncertainty function satisfying rules (1)-(3) is of the form*

$$H(p_1, \dots, p_d) = -c \sum_{i=1}^{d} p_i \log p_i$$

for some $c \geq 0$.

In particular, $E(d) = c \log d$ and $H(p_1, \dots, p_d) = 0$ if $p_i \in \{0, 1\}$ for each i. Assuming that the total number of transmission words is indeed d, then it is a natural to normalize, i.e., take $c = 1/\log d$, or equivalently, to compute logarithms in base d.

Proof. If we break up an equal choice of d^2 possibilities into first d equal possibilities followed by d equal possibilities, we obtain

$$E(d^2) := H(\frac{1}{d^2}, \dots, \frac{1}{d^2}) = H(\frac{1}{d}, \dots, \frac{1}{d}) + \sum_{i=1}^{d} \frac{1}{d} H(\frac{1}{d}, \dots, \frac{1}{d}) = 2E(d).$$

Induction gives $E(d^r) = rE(d)$. Now choose $2 \leq a, b \in \mathbb{N}$ and $r, s \in \mathbb{N}$ such that $a^r \leq b^s < a^{r+1}$. Taking logarithms gives $\frac{r}{s} \leq \frac{\log b}{\log a} \leq \frac{r+1}{s}$. The monotonicity of rule (2) also gives

$$rE(a) = E(a^r) \leq E(b^s) = sE(b),$$

so taking logarithms again, we obtain $\frac{r}{s} \leq \frac{E(b)}{E(a)} \leq \frac{r+1}{s}$. Combining the two, we obtain

$$\left| \frac{E(b)}{E(a)} - \frac{\log b}{\log a} \right| \leq \frac{2}{s}.$$

Since $s \in \mathbb{N}$ can be taken arbitrarily large, it follows that

$$E(b) = c \log b \qquad \text{for } c = \frac{E(a)}{\log a}. \tag{5}$$

The monotonicity of rule (2) implies that $c \geq 0$.

Now assume that $p_i = n_i/N$ for integers n_i and $N = \sum_{i=1}^{d} n_i$. By splitting the choice into N equal possibilities into d possibilities with probability p_i, each of which is split into n_i equal possibilities, by (3), we get

$$E(N) = H(p_1, \dots, p_d) + \sum_{i=1}^{d} p_i E(n_i).$$

Inserting (5), we obtain

$$H(p_1, \dots, p_d) = -c \sum_{i=1}^{d} p_i (\log n_i - \log N) = -c \sum_{i=1}^{d} p_i \log \frac{n_i}{N} = -c \sum_{i=1}^{d} p_i \log p_i.$$

This proves the theorem for all rational choices of $(p_1, \dots, p_d)$. The continuity of rule (1) implies the result for all real probability vectors. □

Suppose we compose messages of n symbols in $\{0, 1\}$, and each symbol has probability p_0 of being a 0 and $p_1 = 1 - p_0$ of being a 1, independently of everything else. Then the bulk of such messages has np_0 zeros and np_1 ones. The exponential growth rate of the number of such words is, by Stirling's formula

$$\begin{aligned} \lim_{n\to\infty} \frac{1}{n} \log \binom{n}{np_0} &= \lim_{n\to\infty} \frac{1}{n} \log \frac{n^n e^{-n} \sqrt{2\pi n}}{(np_0)^{np_0} e^{-np_0} \sqrt{2\pi n p_0}\ (np_0)^{np_0} e^{-np_0} \sqrt{2\pi n p_0}} \\ &= -p_0 \log p_0 - p_1 \log p_1 = H(p_0, p_1). \end{aligned}$$

Exercise 2.4. *Show that you get the same result for the exponential growth rate if $\mathcal{A} = \{1, \dots, d\}$ and the probability of transmitting $a \in \mathcal{A}$ is p_a.*

Recall the convenience of using logarithms base d if the alphabet $\mathcal{A} = \{1, 2, \dots, d\}$ has d letters. In this base, the exponential growth rate is $H(p_1, \dots, p_d) \leq 1$ with equality if and only if all $p_a = 1/d$. Thus the number of the most common words (in the sense of the frequencies of $a \in \mathcal{A}$ deviating very little from p_a) is roughly $d^{nH(p_1,\dots,p_d)}$. This suggests that one could recode the bulk of the possible message with words of length $nH(p_1, \dots, p_d)$ rather than n. Said differently, the bulk of the words $x_1 \dots x_n$ have measure

$$p(x_1, \dots x_n) = \prod_{i=1}^{n} p_{x_i} \approx e^{-nH(p_1,\dots,p_d)}.$$

By the Strong Law of Large Numbers, for all $\varepsilon, \delta > 0$ there is $N \in \mathbb{N}$ such that for all $n \geq N$, up to a set of measure ε, all words $x_1 \dots x_n$ satisfy

$$\left| -\frac{1}{n} \log_d p(x_1 \dots x_n) - H(p_1, \dots, p_d) \right| < \delta.$$

Thus, such δ-typical words can be recoded using at most $n(H(p_1, \dots, p_d) + o(1))$ letters for large n, and the compression rate is $H(p_1, \dots, p_d) + o(1)$ as $n \to \infty$. Stronger compression is impossible. This is the content of Shannon's Source Coding Theorem:

Theorem 2.5. *For a source code of entropy H and a channel with capacity Cap, it is possible, for any $\varepsilon > 0$, to design an encoding such that the transmission rate satisfies*

$$\frac{Cap}{H} - \varepsilon \leq \mathbb{E}(R) \leq \frac{Cap}{H}. \tag{6}$$

No encoding achieves $\mathbb{E}(R) > \frac{Cap}{H}$.

That is, for every $\varepsilon > 0$ there is N_0 such that for very $N \geq N_0$, we can compress a message of N letter with negligible loss of information into a message of $N(H + \varepsilon)$ bits, but compressing it in fewer bit is impossible without loss of information.

Proof. Assume that the source messages are in alphabet $\{1, \dots, d\}$ and letters s_i appear independently with probability p_i, so the entropy of the source is $H = -\sum_i p_i \log p_i$. For the upper bound, assume that the ith letter from the source alphabet require t_i bits to be transmitted.

The expected rate $\mathbb{E}(R)$ should be interpreted as the average number of bits that a bit of a "typical" source message requires to be transmitted. Let $\mathcal{L}_N$ be the collection of N-letter words in the source, and μ_N be the N-fold Bernoulli product measures with probability vector $p = (p_1, \dots, p_d\}$. Let

$$A_{N,p,\varepsilon} = \{s \in \mathcal{L}_N : |\frac{|s|_i}{N} - p_i| < \varepsilon \text{ for } i = 1, \dots, d\}.$$

By the Law of Large Numbers, for any $\delta, \varepsilon > 0$ there is N_0 such that $\mu_N(A_{N,p,\varepsilon}) > 1-\delta$ for all $N \geq N_0$. This suggests that a source message s being "typical" means $s \in A_{N,p,\varepsilon}$, and the transmission length of s is therefore approximately $\sum_i p_i t_i N$. Thus typical words $s \in \mathcal{L}_N$ require approximately $t = \sum_i p_i t_i N$ bits transmission time, and the expected rate is $\mathbb{E}(R) = (\sum_i p_i t_i)^{-1}$.

For the capacity, the number of possible transmissions of t bits is at least the cardinality of $A_{N,p,\varepsilon}$, which is the multinomial coefficient $\binom{N}{p_1 N,\dots,p_d N}$. Therefore, by Stirling's Formula,

$$\begin{aligned}\text{Cap} &\geq \frac{1}{t}\log\binom{N}{p_1 N,\dots,p_d N} \geq \frac{1}{\sum_i p_i t_i N}\log\left((\sqrt{2\pi N})^{1-d}\prod_{i=1}^{d} p_i^{-(p_i N+\frac{1}{2})}\right)\\ &= \frac{-\sum_i p_i \log p_i}{\sum_i p_i t_i} - \frac{\sum_i \log p_i}{2\sum_i p_i t_i N} + \frac{\frac{d-1}{2}\log 2\pi N}{\sum_i p_i t_i N} \geq \mathbb{E}(R) H,\end{aligned}$$

proving the upper bound (with equality in the limit $N \to \infty$).

The coding achieving the lower bound in (6) that was used in Shannon's proof resembled one designed by Fano [9]. It is now known as the Shannon-Fano code and works as follows:

For the lower bound, let again $\mathcal{L}_N$ be the collection of words B of length N in the source, occurring with probability p_B. The Shannon-McMillan-Breiman Theorem implies that for every $\varepsilon > 0$ there is N_0 such that for all $N \geq N_0$,

$$|-\frac{1}{N}\log p_B - H| < \varepsilon \text{ for all } B \in \mathcal{L}_N \text{ except for a set of measure } < \varepsilon.$$

Thus the average

$$G_N := -\frac{1}{N}\sum_{B\in\mathcal{L}_N} p_B \log p_B \to H \qquad \text{as } N \to \infty.$$

If we define the conditional entropy of symbol a in the source alphabet following a word in $\mathcal{L}_N$ as

$$F_{N+1} = H(Ba|B) = -\sum_{B\in\mathcal{L}_N}\sum_{a\in\mathcal{S}} p_{Ba}\log_2\frac{p_{Ba}}{p_B},$$

then after rewriting the logarithms, we get (using telescoping series) $F_{N+1} = (N+1)G_{N+1} - NG_N$, so $G_N = \frac{1}{N}\sum_{n=0}^{N-1} F_{n+1}$. The conditional entropy is decreasing as the words B get longer. Thus F_N is decreases in N and G_N is a decreasing sequence as well.

Assume that the words $B_1, B_2, \dots, B_n \in \mathcal{L}_N$ are arranged such that $p_{B_1} \geq p_{B_2} \geq \dots \geq p_{B_n}$. Shannon encodes the words B_i in binary as follows. Let $P_s = \sum_{i<s} p_{B_i}$, and choose $m_s = \lceil -\log p_{B_s}\rceil$, encode B_s as the first m_s digit of the binary expansion of P_s, see Table 1. Because $P_{s+1} \geq P_s + 2^{-m_s}$, the encoding of B_{s+1} differs by at least one in the digits of the encoding of B_s. Therefore all codes are different.

The average number of bits **per symbol** is $H' = \frac{1}{N}\sum_s m_s p_{B_s}$, so

$$G_N = -\frac{1}{N}\sum_s p_{B_s}\log p_{B_s} \le H' < -\frac{1}{N}\sum_s p_{B_s}(\log p_{B_s} - 1) = G_N + \frac{1}{N}.$$

Therefore the average rate of transmission is

$$\frac{\text{Cap}}{H'} \in \left[\frac{\text{Cap}}{G_N + \frac{1}{N}}\ ,\ \frac{\text{Cap}}{G_N}\right].$$

Since G_N decreases to the entropy H, the above tends to Cap $/H$ as required.

p_{B_s}	P_s	m_s	Shannon	Fano
$\frac{8}{36}$	$\frac{28}{36}$	3	110	11
$\frac{7}{36}$	$\frac{21}{36}$	3	101	101
$\frac{6}{36}$	$\frac{21}{36}$	3	011	100
$\frac{5}{36}$	$\frac{15}{36}$	3	010	011
$\frac{4}{36}$	$\frac{6}{36}$	4	0010	010
$\frac{3}{36}$	$\frac{3}{36}$	4	0001	001
$\frac{2}{36}$	$\frac{1}{36}$	5	00001	0001
$\frac{1}{36}$	$\frac{0}{36}$	6	00000(0)	0000

Table 1: An example of encoding using Shannon code and Fano code.

Fano [9] used a different and slightly more efficient encoding, but with the same effect (the difference negligible for large values of N). He divides $\mathcal{L}_N$ into two groups of mass as equal to 1/2 as possible. The first group gets first symbol 1 in its code, the other group 0. Next divide each group into two subgroups of mass as equal to $1/2\times$ probability of the group as possible. The first subgroups get second symbol 1, the other subgroup 0, etc. See Table 1. $\square$

2.2 Data Compression over Noisy Channels

Shannon's Source Code Theorem 2.5 extends to noisy transmission channels, i.e., channels through which a certain percentage of the transmissions arrive in damaged form. The only thing to changed in the statement of the theorem is the definition of capacity. For example, imagine a binary message, with symbol probabilities $_0$ and $p_1 = 1 - p_0$, is transmitted and a fraction q of all the symbols is distorted from 0 to 1 and vice versa. This means that symbol i in the received signal y has a chance

$$\mathbb{P} = \mathbb{P}(x = i | y = i) = p_i(1-q) + (1-p_i)q$$

$$\begin{tikzcd}
0 \arrow[r, "1-q"] \arrow[dr, "q"] & 0 \\
1 \arrow[r, "1-q"'] \arrow[ur, "q"'] & 1
\end{tikzcd} \tag{7}$$

in the sent signal x. If $q = \frac{1}{2}$, then $\mathbb{P} = \frac{1}{2}$, so every bit of information will be transmitted with total unreliability. But also if q is small, $\mathbb{P}$ can be very different from $\mathbb{P}(y = i)$. For example, if $p_0 = q = 0.1$, then $\mathbb{P} = 0.1(1 - 0.01) + (1 - 0.1)0.1 = 0.18$. The key notion to measure this uncertainty of sent symbol is the **conditional entropy** of x given that y is received:

$$H(x|y) = -\sum_{i,j} \mathbb{P}(x_i \wedge y_j) \log \frac{\mathbb{P}(x_i \wedge y_j)}{\mathbb{P}(y_j)}$$

were the sum is over all possible sent message x_i and received messages y_j. This uncertainty $H(x|y)$ is called the **equivocation**. If there is no noise, then knowing y gives full certainty about x, so the equivocation $H(x|y) = 0$, but also $q = 1$, i.e., every symbol is received distorted, knowing y gives full knowledge of x and $H(x|y) = 0$. In the above example:

$$\begin{aligned} H(x|y) \quad = \quad & -p_0(1-q)\log_2 \frac{p_0(1-q)}{p_0(1-q)+p_1 q} - p_1 q \log_2 \frac{p_0 q}{p_0(1-q)+p_1 q} \\ & -p_1(1-q)\log_2 \frac{p_1(1-q)}{p_0 q + p_1(1-q)} - p_0 q \log_2 \frac{p_0 q}{p_0 q + p_1(1-q)}. \end{aligned} \tag{8}$$

The actual information transmitted, known as the **mutual information**, is defined as

$$I(X|Y) = H(x) - H(x|y)$$

In, say, the binary alphabet, we can interpret $2^{H(x)n+o(n)}$ as the approximate possible number of length n source messages, up to a set of measure ε, that tends to with an error that tends to zero as $n \to \infty$. For each source message x, the number of received messages y generated from x via transmission errors is $2^{H(y|x)n+o(n)}$, see Figure 13. Analogous interpretations hold for $2^{H(y)n+o(n)}$ and $2^{H(x|y)n+o(n)}$.
The total number of non-negligible edges in Figure 13 is therefore approximately

$$2^{H(x)n+o(n)} \cdot 2^{H(y|x)n+o(n)} \approx 2^{H(y)n+o(n)} \cdot 2^{H(x|y)n+o(n)} \approx 2^{H(x \vee y)n+o(n)}.$$

Taking logarithms, dividing by $-n$, taking the limit $n \to \infty$ and finally adding $H(x) + H(y)$ gives the helpful relations

$$I(X|Y) = H(x) - H(x|y) = H(y) - H(y|x) = H(x) + H(y) - H(x \vee y). \tag{9}$$

If the noiseless channel capacity $\text{Cap}_{\text{noiseless}}$ is less than the equivocation, then it is impossible to transmit the message with any reliable information retained. Simply, uncertainty is produced faster than the channel can transmit. If the equivocation is less than the capacity, then by adding (otherwise superfluous) duplicates of the message, or control symbols, the message can be transmitted such that it can be reconstructed afterwards if negligible errors. However, the smaller the difference $\text{Cap}_{\text{noiseless}} - H(x|y)$,

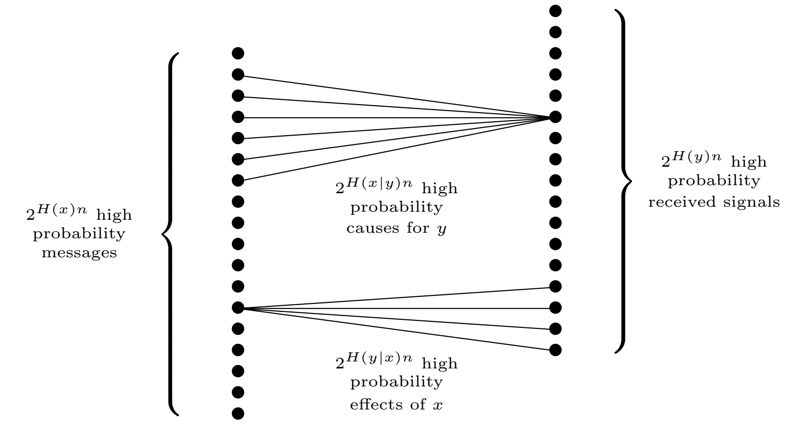

$2^{H(x)n}$ high probability messages
$2^{H(x|y)n}$ high probability causes for y
$2^{H(y)n}$ high probability received signals
$2^{H(y|x)n}$ high probability effects of x

Figure 13: Interpreting $2^{H(x)n}$, $2^{H(y)n}$, $2^{H(x|y)n}$ and $2^{H(y|x)n}$

the more duplicates and/or control symbols have to be send to allow for reconstruction. It turns out that the correct way to define the capacity of a noisy channel is

$$\mathrm{Cap}_{\text{noisy}} = \max_x I(X|Y) = \max_x H(x) - H(x|y)$$

where the maximum is taken over all possible source messages only, because the distribution of the received messages can be derived from x and knowledge to what extent the channel distorts messages. The distribution of x that achieves the maximum is called the **optimal input distribution**. To compute $\mathrm{Cap}_{\text{noisy}}$ in the example of (8) we need to maximize $H(x) - H(x|y)$ over p_0 (because $p_1 = 1 - p_0$ and q is fixed). Due to symmetry, the maximum is achieved at $p = \frac{1}{2}$, and we get

$$\mathrm{Cap}_{\text{noisy}} = \log_2 2 + q \log_2 q + (1-q)\log_2(1-q). \tag{10}$$

This confirms full capacity $1 = \log_2 2$ if there is no noise and $\mathrm{Cap}_{\text{noisy}} = 0$ if $q = \frac{1}{2}$.

Let $Q = (q_{ij})$ be the probability matrix where q_{ij} stands for the probabiity that symbol $i \in \mathcal{S}$ is received as $j \in \mathcal{A}$. Thus if $\mathcal{S} = A$, then q_{ii} is the probability that symbol i is transmitted correctly.

Proposition 2.6. *Assuming that Q is an invertible square matrix with inverse $Q^{-1} = (q_{ij}^{-1})$, the optimal probability for this noisy channel is*

$$p_i = \sum_t q_{ti}^{-1} \exp\left(-\, Cap_{noisy} \sum_s q_{ts}^{-1} + \sum_{s,j} q_{ts}^{-1} q_{sj} \log q_{sj} \right),$$

where the noisy capasity Cap_{noisy} should be chosen such that $\sum_i p_i = 1$.

Proof. We maximize

$$\begin{aligned} I(X|Y) &= -\sum_i p_i \log p_i + \sum_{i,j} p_i q_{ij} \log \frac{p_i q_{ij}}{\sum_k p_k q_{kj}} \\ &= \sum_{i,j} p_i q_{ij} \log q_{ij} - \sum_{i,j} p_i q_{ij} \log \sum_k p_k q_{kj} \end{aligned}$$

over p_i subject to $\sum_i p_i = 1$ using Lagrange multipliers. This gives

$$\sum_j q_{sj} \log \frac{q_{sj}}{\sum_k p_k q_{kj}} = \mu \qquad \text{for all } s \in \mathcal{S}. \tag{11}$$

Multiply the equations (11) with p_s (with $\sum_s p_s = 1$) and sum over s to get

$$\mu = \sum_{s,j} p_s q_{sj} \log \frac{q_{sj}}{\sum_k p_k q_{kj}} = \mathrm{Cap}_{\mathrm{noisy}}\,.$$

Now multiply (11) with q_{ts}^{-1}, sum over s and take the exponential. This gives

$$\sum_k p_k q_{kt} = \exp\left(\sum_{s,j} q_{ts}^{-1} q_{sj} \log q_{sj} - \mathrm{Cap}_{\mathrm{noisy}} \sum_s q_{ts}^{-1} \right).$$

Therefore

$$p_i = \sum_t q_{ti}^{-1} \exp\left(\sum_{s,j} q_{ts}^{-1} q_{sj} \log q_{sj} - \mathrm{Cap}_{\mathrm{noisy}} \sum_s q_{ts}^{-1} \right),$$

as claimed. □

Remark 2.7. *Suppose that the matrix Q has a diagonal block structure, i.e., the source alphabet can be divided into groups g with symbols in separate groups never mistaken for one another. Then $Cap_{noisy} = \log_2 \sum_g 2^{Cap_g}$, where Cap_g is the noisy capacity of the group g. The optimal probability for all symbols in group g together is $\mathbb{P}_g = 2^{Cap_g} / \sum_{g'} 2^{Cap_{g'}}$.*

Exercise 2.8. *The noisy typewriter is a transmission channel for the alphabet $\{a, b, c, \dots, z, -\}$, where $-$ stands for the space. Imagine a circular keyboard on which typing a letter results in that latter or to one of its neighbors, all three with probability $1/3$.*

1. *Compute the capacity Cap_{noisy} of this channel and an optimal input distribution.*
2. *Find an optimal input distribution so that the received message can be decoded without errors.*

Going back to Figure 13, assume that among the most likely transmissions, we choose a maximal subcollection that have disjoint sets of most likely received messages. The cardinality of this subcollection is approximately

$$\frac{2^{H(y)n}}{2^{H(y|x)n}} = 2^{(H(y)-H(y|x))n} \leq 2^{n\,\mathrm{Cap}_{\mathrm{noisy}}}.$$

Then such received messages can be decoded with a negligible amount of error. Maximizing over all input distributions, we obtain that a message can be transmitted virtually error-free at a rate of C bits per transmitted bit. This is a heuristic argument for Shannon's Noisy Source Code Theorem.

Theorem 2.9. *For transmission of a message of entropy H through a noisy channel with capacity Cap_{noisy} and any $\varepsilon > 0$, there is $N_0 \in \mathbb{N}$ such that every message of length $N \geq N_0$ can be transmitted through the channel at expected rate $\mathbb{E}(R)$ arbitrarily close to $\frac{Cap_{noisy}}{H}$ such that the proportion of errors is at most ε.*

If we allow a proportion of errors δ, the expected rate for this proportion of errors is $\mathbb{E}(R(\delta))$ can be made arbitrarily close to $\frac{Cap_{noisy}}{H(1-h_2(\delta))}$, for the entropy function $h_2(\delta) = -\delta \log_2 \delta - (1-\delta)\log_2(1-\delta)$. Faster rates at this proportion of errors cannot be achieved.

Proof. We will give the proof for the binary channel of (7), so the channel capacity is $\text{Cap}_{\text{noisy}} = 1 + q\log_2 q + (1-q)\log_2(1-q) =: 1 - h_2(q)$ as in (10). This gives no loss of generality, because we can always recode the source (of entropy H) into binary in an entropy preserving way[3]. By the Noiseless Source Code Theorem 2.9, the rate "gains" a factor $1/H$ because the compression from the source alphabet into the binary alphabet occurs at a noiseless capacity $\text{Cap} = \lim_t \frac{1}{t}\log 2^t = 1$. This factor we have to multiply the rate with at the end of the proof.

Also we will use linear codes such as Hamming codes[4] which postdate Shannon's proof, but shows that already an elementary linear code suffices to achieve the claim of the theorem.

Assume that the source messages have length K, which we enlarge by M parity check symbols to a source code x of length $N = K + M$. This is done by a Hamming code in the form of an $M \times N$ matrix $L \in \mathbb{F}_2^{M\times N}$ of which the right $M \times M$ submatrix is the parity check matrix. When x is transmitted over the noisy binary channel, approximately Nq errors appear, i.e., Nq symbols are flipped, and the received word is $y = x + w$, where w stands for the noise. It is a sample of N Bernoulli trials with success (i.e., error) chance q, occurring with probability $q^{|w|_1}(1-q)^{|w|_0}$.

For an arbitrary $\varepsilon > 0$, there is some constant C_ε such that the noise words w with $|w|_1 > (N + C_\varepsilon\sqrt{N})q$ have total probability $< \varepsilon/2$. The number of noise words w with $|w|_1 \leq (N + C\sqrt{N})q$ is $\leq 2^{(N+C_\varepsilon\sqrt{N})h_2(q)}$.

The choice of a Hamming code defines the actual transmission, so we can decide to choose x according to the optimal input distribution. This will give an output distribution satisfying $H(y) - H(y|x) = \text{Cap}_{\text{noisy}}$, and we can select a subset of "typical" outcomes Y_{typ} as those produce from some source x and a non-typical noise w. The probability of not being in Y_{typ} is less than $\varepsilon/2$.

There are altogether 2^M syndromes, i.e., outcomes of $S(y) = yL^T \in \{0,1\}^M$. The Hamming encoding satisfies $S(x) = 0$, so $S(y) = S(x + w) = S(w)$. The sets $\{y \in Y_{\text{typ}}$:

[3]Note that we may have to use blocks of 0s and 1s instead of single symbols if the entropy is larger than $\log 2$.

[4]For this proof we would like to acknowledge the online lectures by Jacob Foerster (Cambridge University) https://www.youtube.com/watch?v=KSV8KnF38bs, based in turn on the text book [18].

$S(y) = z\}$ are the 2^M disjoint subsets of Y_{typ} from which x can be reconstructed, by subtracting the coset leader of $S(y)$ from y. This is error-free except (possibly) if

1. $y \notin Y_{\text{typ}}$, but this happens with probability $\leq \varepsilon/2$, independently of the choice of Hamming encoding;

2. there is some other $\tilde{y} = \tilde{x} + \tilde{w} \in Y_{\text{typ}}$ such that $S(\tilde{y}) = S(y)$, i.e., $S(\tilde{w} - w) = 0$.

The probability that the latter happens

$$\sum_{y=x+w \in Y_{\text{typ}}} \mathbb{P}(w) 1_{\{\exists\ \tilde{w} \neq w\ \tilde{y} \in Y_{\text{typ}} (\tilde{w}-w)L^T = 0\}} \leq \sum_{w} \mathbb{P}(w) \sum_{\tilde{w} \neq w} 1_{\{(\tilde{w}-w)L^T = 0\}}.$$

is difficult to compute. However, we can average over all 2^{M^2} possible $M \times N$-matrices L, occurring with probability $\mathbb{P}(L)$. This gives an average failure probability less than

$$\sum_{L} \mathbb{P}(L) \sum_{w} \mathbb{P}(w) \sum_{\tilde{w} \neq w} 1_{\{(\tilde{w}-w)L^T = 0\}} \leq \sum_{w} \mathbb{P}(w) \sum_{\tilde{w} \neq w} \sum_{L} \mathbb{P}(L) 1_{\{(\tilde{w}-w)L^T = 0\}}.$$

The inner sum, however, equals 2^{-M} because the probability of any entry of $(\tilde{w} - w)L^T$ being zero is $1/2$, independent of all other entries. Since there are $2^{(N+C_\varepsilon\sqrt{N})h_2(q)}$ possible noises $\tilde{w}$ such that $\tilde{y} \in Y_{\text{typ}}$, the *average* failure probability over all Hamming codes is $\leq 2^{(N+C_\varepsilon\sqrt{N})h_2(q)} 2^{-M}$. Assuming that $N(h_2(q) + 2\varepsilon) > M \geq N(h_2(q) + \varepsilon) > N$, this probability is $\leq 2^{-N\varepsilon} < \varepsilon/2$. Therefore the failure probability averaged over all Hamming codes is at most ε, and therefore there must exist at least one (probably many) Hamming code that achieves the required failure rate. But $h_2(q) + 2\varepsilon > M/N \geq h_2(q) + \varepsilon$ implies that the transmission rate for this Hamming code satisfies

$$\text{Cap}_{\text{noisy}} - 2\varepsilon < \frac{K}{N} = 1 - \frac{M}{N} = 1 - h_2(q) - \varepsilon \leq \text{Cap}_{\text{noisy}} - \varepsilon,$$

as claimed.

For the second statement, that is, if we allow a proportion δ of errors, we use the Hamming code of the first part of the proof in reverse. We chop the source message into blocks of length N and consider each block as a code word of which the last M bits play the role of parity check symbols (although they really aren't) and the first K bits play the role of actual information. For typical messages (i.e., all up to an error ε of all the possible blocks in $\mathcal{L}_N$), we can use a Hamming code for which $M = \lceil h_2(\delta)N \rceil$, and then we throw these symbols simply away. We choose the initial block size N that he remaining $K = N - M = N(1 - h_2(\delta))$ bits are at the capacity of the noisy channel, i.e., these K bits take $\text{Cap}_{\text{noisy}} K$ bits to transmit, virtually error-free. But then a **typical** original N bits message takes $\frac{\text{Cap}_{\text{noisy}}}{1-h_2(\delta)} N$ bits to transmit at an error proportion δ, so the expected noisy rate $\mathbb{E}(R(\delta)) = \frac{\text{Cap}_{noisy}}{1-h_2(\delta)}$, proving the second part.

Let us now argue that the rate R cannot exceed the capacity $\text{Cap}_{\text{noisy}}$, following [22]. We will use a code of n-letter code words on a two-letter alphabet; the loss of generality in this choice becomes negligible as $nto\infty$. There are thus 2^{nR} code words among 2^n n-letter words in total, and for this rate th code is optimally used if all the code words have probability 2^{-nR} of occurring, and hence the entropy of code words X^n chosen according uniform distribution is $H(X^n) = nR$. The received message Y^n is also an n-letter word, and since X^n is sent letter by letter, with errors occurring independently in each letter,

$$\mathbb{P}(Y^n|X^n) = \mathbb{P}(y_1 y_2 \dots y_n | x_1 x_2 \dots x_n) = \prod_{i=1}^{n} \mathbb{P}(y_i|x_i).$$

Hence the conditional entropy, being the average of minus the logarithm of the above quantity, satisfies

$$H(Y^n|X^n) = -\sum_{i=1}^{n} \mathcal{P}(y_i|x_i) \log \mathbb{P}(y_i|x_i) = \sum_{i=1}^{n} H(uy_i|x_i).$$

Entropy is subadditive, so the mutual information is

$$I(Y^n|X^n) = H(Y^n) - H(Y^n|X^n) \leq \sum_{i-=1}^{n} H(y_i) - H(y_i|x_i) = \sum_{i=1}^{n} I(y_i|x_i) \leq n \operatorname{Cap}_{\text{noisy}}, \tag{12}$$

because $\operatorname{Cap}_{\text{noisy}}$ is the mutual information maximized over all input distributions. By the symmetry of (9), also

$$I(Y^n|X^n) = H(X^n) - H(X^n|Y^n) = nR - H(X^n|Y^n) = \sum_{i=1}^{n} I(y_i|x_i) \leq n \operatorname{Cap}_{\text{noisy}}, \tag{13}$$

Reliably decoding the received message means that $\frac{1}{n} H(X^n|Y^n) \to 0$ as $n \to \infty$. Hence, combining (13) and (12) gives $R \leq \operatorname{Cap}_{\text{noisy}} + o(1)$ as $n \to \infty$. This gives the upper bound $R \leq \operatorname{Cap}_{\text{noisy}}$. $\square$

2.3 Symbol Codes

Shannon's Source Code Theorem 2.5 gives the theoretical optimal bounds for encoding messages in the shortest way. Suppose that a source $\mathcal{S}$ is an ensemble from which the elements $s \in \mathcal{S}$ are produced with frequencies p_s, so the source entropy is $H(\mathcal{S}) = \sum_{s \in \mathcal{S}} p_s \log p_s$. Translating Shannon's Theorem, assuming that the channel is simple such that the capacity is $\log_d d = 1$, for every $\varepsilon > 0$ there exists a code that turns a string $s_1 \dots s_n$ of sampled symbols such into a code word $c(s_1 \dots s_n)$ where

$$nH(\mathcal{S}) \leq |c(s_1 \dots s_n)| \leq n(H(\mathcal{S}) + \varepsilon) \qquad \text{as } n \to \infty, \tag{14}$$

and no code performs better (without losing information) that $nH(\mathcal{S})$. What are practical codes that achieve this? That is, an code c that:

1. turns every string $s_1 \dots s_n$ from the source into a string $a_1 \dots a_k$ in the code alphabet $\mathcal{A}$;
2. is uniquely **decodable** (or **lossless**, i.e., can be inverted;
3. achieves (14) with ε as small as possible;
4. is easily computable, i.e., cheap, quick and preferably without the use of a huge memory.

The first thing to try is a **symbol code**, i.e, c assigns a string $c(s) \in \mathcal{A}^*$ of to each $s \in \mathcal{S}$, and extend this by concatenation. That is, a symbol code is a substitution, except that it is not necessarily stationary. The rule encode s_i in the source string is allowed to depend on the context, or rather the history $s_{i-1}, s_{i-2}, \dots$

Definition 2.10. *A* **prefix (suffix) code** *is a code c such that no code word $c(s)$ is a prefix (suffix) of any other code words $c(s')$.*

It is easy to verify that Shannon code and Fano code are both prefix (but not suffix) codes.

Lemma 2.11. *Every prefix and suffix code is uniquely decodable.*

Proof. Suppose c is a prefix code and $c(s_1 \dots s_n) = a_1 \dots a_k$. Start parsing from the left, until you find the first $a_1 \dots a_j = c(s)$ for some $s \in \mathcal{S}$. Since $c(s)$ is not the prefix of any other $c(s')$, we must have $s = s_1$. Continue parsing from symbol a_{j+1}, etc. For suffix codes we do the same, only parsing from right to left. □

Since it is uncommon to parse from right to left (in the direction messages are transmitted), we always prefer prefix codes over suffix codes. In this, decodable is different from the notion recognizable in substitution shift, because decoding algorithms cannot depend on future digits. Any prefix code can be represented as an edge-labeled subtree of the binary tree (or d-adic tree if $d = \#\mathcal{A}$) in which each code word is the label of a path from the root to a leaf (i.e, a non-root vertex that has only one edge connected to it), see Figure 14.

The following condition, called the **Kraft inequality**, is a necessary condition for a symbol code to be uniquely decodable.

Proposition 2.12. *A uniquely decodable code with codewords $c(s)$ in alphabet $\mathcal{A}$ with $d = \#\mathcal{A}$ satisfies*

$$\sum_{s \in \mathcal{S}} d^{-|c(s)|} \leq 1, \tag{15}$$

Proof. Let $d = \#\mathcal{A}$ and $\ell_s = |c(s)|$, so $1 \leq \ell_s \leq \ell_{\max}$ for the maximum code word length $\ell_{\max}$. Let $K = \sum_s d^{-\ell_s}$, and therefore

$$K^n = \left(\sum_s d^{-\ell_s} \right)^n = \sum_{s_1=1}^{\ell_{\max}} \cdots \sum_{s_n=1}^{\ell_{\max}} d^{-(\ell_{s_1} + \cdots + \ell_{s_n})}$$

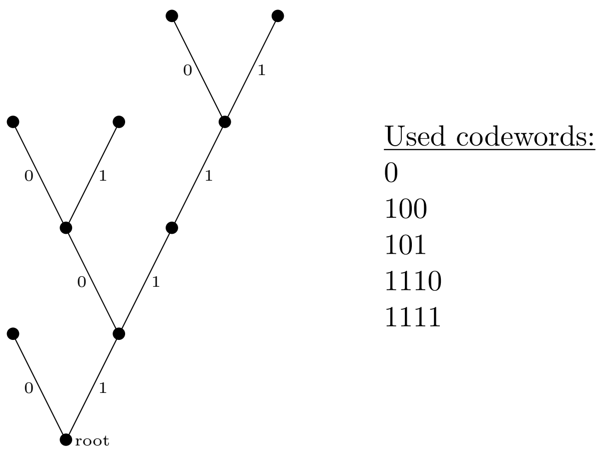


Figure 14: An example of a tree representing prefix code.

But each word is uniquely decodable, so to every string x of length $\ell \leq n\ell_{\max}$, there is at most one choice of $s_1, \dots, s_n$ such that $x = c(s_1) \dots c(s_n)$. Since there at at most n^ℓ words of this length, we get

$$K^n \leq \sum_{\ell=n}^{n\ell_{\max}} d^\ell d^{-\ell} \leq n\ell_{\max}.$$

Taking the n-th root on both sides and th limit $n \to \infty$, we get $K \leq 1$. □

If equality holds in (15), then we say that the code is **complete**. For a complete code, the average frequencies of each $a \in \mathcal{A}$ must be $1/d$ (and there cannot be correlation between digits), because otherwise the entropy of the encoding is not maximal and we could encode it further.

Prefix/suffix codes that fill the subtree entirely are complete. Indeed, such a code has $d-1$ words of length k and d words of the maximal length n, so

$$\sum_s d^{-c(s)} \leq (d-1) \sum_{k=1}^{n-1} d^{-k} + d \cdot d^{-n} = 1.$$

Every unique decodable code c is equivalent to a prefix code. To see this, order $s \in \mathcal{S}$ according to their code word lengths, and then recode them lexicographically, maintaining code length, and avoiding prefixes. Due to the Kraft inequality, there is always space for this.

For the next result, we need the **Gibbs inequality**

Lemma 2.13. *If (p_s) and (q_s) are probability vectors, then the Gibbs inequality*

$$\sum p_s \log_d p_s/q_s \geq 0 \tag{16}$$

holds with equality only if $q_s = p_s$ for all s.

Proof. Indeed, since $\log x \leq x - 1$ for all $x > 0$ (with equality only if $x = 1$), we have

$$\begin{aligned}\sum_s p_s \log_d \frac{p_s}{q_s} &= \frac{-1}{\log d} \sum_s p_s \log \frac{q_s}{p_s} \\ &\geq \frac{-1}{\log d} \sum_s p_s (\frac{q_s}{p_s} - 1) = \frac{1}{\log d}(1 - \sum_s q_s) = 0,\end{aligned}$$

with equality only if $p_s = q_s$ for all s. □

Theorem 2.14. *Let $\mathcal{S}$ be an ensemble with probabilities p_s for $s \in \mathcal{S}$. Suppose that c is a prefix code such that $|c(s)| = \lceil \log 1/p_i \rceil$. Then the expect length of c satisfies*

$$H(\mathcal{S}) \leq \mathbb{E}(L) \leq H(\mathcal{S}) + 1,$$

and no symbol code has a shorter expected length that $H(\mathcal{S})$.

Proof. Let $q_s = d^{-|c(s)|}/Z$, where $d = \#\mathcal{A}$ is the alphabet size, and $Z := \sum_s d^{-|c(s)|}$ is the normalizing factor. By the Kraft inequality $Z \leq 1$, with equality if the code is complete. Now the expected length of the code word is

$$\begin{aligned}\sum_s p_s |c(s)| &= \sum_s p_s \log_d \frac{1}{q_s} = -\sum_s p_s \log_d p_s + \sum_s p_s \log_d \frac{p_s}{q_s} - \log Z \\ &= H(\mathcal{S}) \quad + \quad \underbrace{\sum_s p_s \log_d \frac{p_s}{q_s}}_{\geq 0 \text{ by Gibbs ineq.}} \quad \underbrace{- \quad \log Z}_{\geq 0 \text{ by Kraft ineq.}} \geq H(\mathcal{S}),\end{aligned}$$

with equality only if the code is complete with codeword lengths equal to $-\log_d p_s$. Conversely, if we make a code such that $|c(s)| = \lceil \log_d 1/q_s \rceil$, then

$$\sum_s p_s |c(s)| \leq \sum_s p_s (\log_d \frac{1}{p_s} + 1) = H(\mathcal{S}) + 1,$$

as required. □

Hence this coding exceeds the theoretical lower bound by not more than 1. For very short codes, for instance if every letter of the message is encoded separately, this +1 is relatively large compared to the entropy. Also symbol codes do not compress at all if the alphabet of he source and the encoding are the same. One solution is therefore to encode entire blocks B, say of length n, of message letters, each according to their frequency of occurrence. The corresponding entropy is

$$\begin{aligned}H(\mathcal{S}^n) &= -\sum_{s_1,\dots,s_n \in \mathcal{S}} p_{s_1} \cdots p_{s_n} \log p_{s_1} \cdots p_{s_n} \\ &= -\sum_{s_1,\dots,s_n \in \mathcal{S}} p_{s_1} \cdots p_{s_n} (\log p_{s_1} + \cdots + \log p_{s_n}) \\ &= -\left(\sum_{s_1} p_{s_1} \log p_{s_1} + \cdots + \sum_{s_1} p_{s_n} \log p_{s_n} \right) = nH(\mathcal{S}).\end{aligned}$$

In this way we achieve

$$\frac{1}{n}\sum_{s\in\mathcal{S}} p_s \ell_s \leq \frac{1}{n}(H(\mathcal{S}^n)+1) = H(\mathcal{S}) + \frac{1}{n}.$$

and by taking n large, we can get arbitrarily close to $H(\mathcal{S})$. A disadvantage of block codes, however, is that they tend to use a lot of memory.

The **Huffman code** [14] is a prefix code that achieves (14) and it has a very simple algorithm to produce it, see Figure 15. We construct a tree starting with $\#\mathcal{S}$ leaves, i.e., vertices labeled with p_s. Take the two vertices with the smallest probabilities and connected each of them with a new vertex with label the sum of these probabilities. Repeat this rule with the yet unconnected vertices and the newly created vertices, until all vertices are connected in a single tree, and its root has the total probability as label.

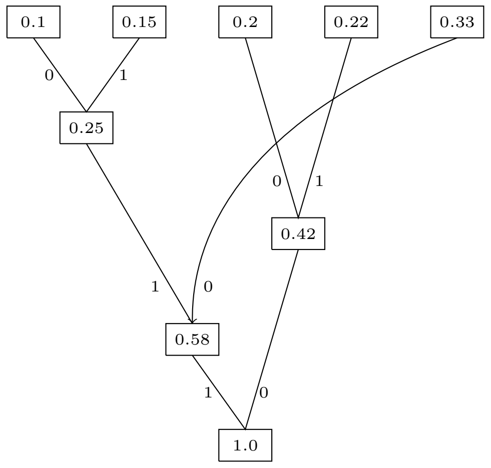


Figure 15: An edge-labeled tree illustrating the Huffman algorithm.

Huffman codes are used in PKzip, JPEG and MP3 encryption algorithms. It widespread use is aided by its simplicity, and the fact that, contrary to several other data compression algorithms, its patent ran out in 2010.

Proposition 2.15. *The Huffman code achieves* (14)*; in fact, it performs at least as good as any symbol code.*

Proof. Suppose by contradiction that the optimal code c id different from and has shorter expected code length than the Huffman code h. Without loss of generality, we can assume that c is a prefix code, and that there are at least two symbols in $\mathcal{S}$ with the same longest code length. Let $s, s' \in \mathcal{S}$ have the smallest frequencies p_s ad ps'. In the Huffman code, $|h(s)| = |h(s')|$. Suppose that $|c(s)| < |c(s')|$, then there is yet a third $s'' \in \mathcal{S}$ with larger frequency $p_{s''}$ such that $|c(s'')| \geq |c(s')| > |c(s)|$. Thus swapping the code words of s and s'' make c more efficient, contradicting that c is already the most efficient code. Thus $|c(s)| = |c(s')|$.

Replace s and s' with a single symbol with frequency $p_s + p_{s'}$ and repeat the argument. □

At any moment in the encoding of the string, say at symbol s_i, the Huffman code can be easily adapted to changes in the probability vector $(p_s)_{s\in\mathcal{S}}$. For example,

- if the precise vector $(p_s)_{s\in\mathcal{S}}$ is unknown, it can be estimated by the occurrence frequencies of the symbols $s \in \mathcal{S}$ in the message up to symbol s_k;
- we can let it depend on the previously read symbol. After all, the letter 'q' is almost always followed by a "u', and vowels are more likely to follow consonants and vice versa. Such ideas of letting the letter frequency depend on the symbols read in a previous window is called **prediction by partial matching (PPM)**. It can easily be combined with other compression algorithms too.

For the case that the frequency of letters $s \in \mathbb{S} = \{0, \dots, N\}$ has a geometric distribution with $p_s = 2^{-s+1}$ for $0 < s < N$ and $p_N = 2^{-N}$, then the Huffman code reduces to the **unary code**, i.e., $c(s) = 1^s0$ for $0 < s < N$ and $c(N) = 1^N$. In 1966, Solomon Golomb [10] used this fact to create a code that works best for letter frequencies p_s that are geometric districally distributed with some rate ρ, but not necessarily equal to 1/2. However, because the computer use of base 2, this base is not simply replace by $\lceil 1/r \rceil$. Instead, if ρ is close to $2^{-1/m}$ for some integer m, it uses use m different code words with the same length and unary prefix. Take $k \in \mathbb{N}$ such that $2^{k-1} < m \le 2^k$, and set $s = qm + r$. Then q is encoded in a unary way (with zero at the end) and the remainder r is encoded roughly in binary. The detailed algorithm is

1. $q = \lfloor s/m \rfloor$; $r = q \pmod{m}$; $k = \lfloor \log_2 m \rfloor$;
2. Start with code 1^q0;
3. If $r \le 2^k - m$ then encode r in binary and attach to 1^q0;
   Otherwise encode $r + 2^k - m$ in binary and attach to 1^q0.

In the 1970s, Robert Rice [23] picked up the idea again. His encoding coincides with Golomb encoding, for $m = 2^k$ only. Indeed, if $m = 2^k$, then line 3. in the above algorithm reduces to: encode r in binary and attach to 1^q0. In table form:

Golomb	m = 1	m = 2	m = 3	m = 4	m = 5	m = 6	m = 7	m = 8
Rice	k = 0	k = 1		k = 2				k = 3
s = 0	0	00	00	000	000	000	000	0000
1	10	01	010	001	001	001	0010	0001
2	110	100	011	010	010	0100	0011	0010
3	1110	101	100	011	0110	0101	0100	0011
4	11110	1100	1010	1000	0111	0110	0101	0100
5	111110	1101	1011	1001	1000	0111	0110	0101
6	1111110	11100	1100	1010	1001	1000	0111	0110
7	11111110	11101	11010	1011	1010	1001	1000	0111
8	111111110	111100	11011	11000	10110	10100	10010	10000

Golomb-Rice encoding is used in several audio-encoding systems, such as MPEG-4 ALS.

2.4 Arithmetic Coding

In a way, **arithmetic coding** is an application of Lochs' Theorem (which in turn is an application of the Shannon-McMillan-Breiman Theorem, but this connection is rarely flagged up. It was invented by Elias, and further described in [21, 24, 25, 30][5]. Arithmetic coding works well with any source or coding alphabet, is adapted to changing probability vectors $(p_s)_{s \in \mathcal{S}}$ and gives results extremely close to the theoretic limit. It works by encoding strings instead of letters, and the performance is independent on the string length, so using longer strings, or directly the whole message, is no problem, maybe even preferable. The downside is that arithmetic coding requires extreme computation precision, because it needs to compute real numbers with excessive accuracy.

The idea is as follows. Divide the unit interval into $\#\mathcal{S}$ intervals J_{s_1}, labeled by the letters $s_1 \in \mathcal{S}$ and of lengths p_{s_1}. Then each interval s_1 is divided in into $\#\mathcal{S}$ intervals, labeled as $s_1 s_2$, $s_2 \in \mathcal{S}$, and of relative lengths p_{s_2}, that is: actual length $p_{s_1} p_{s_2}$. Continuing this way for n steps, we have a very fine partition into intervals, labeled by strings of length n. In practice, we only need to compute the interval J of the string $s_1 \dots s_n$ we want to encode, and this interval has length $|J| = p_{s_1} \cdots p_{s_n}$. Now find the largest dyadic interval D (or d-adic if $\#\mathcal{A} = d$) that is contain in J. This interval is labeled by a string $a_1 \dots a_k \in \mathcal{A}^k$ (which indicate the first k digits (base d) of the real numbers contained in D. This string $a_1 \dots a-_k = c(s_1 dotss_n)$ is the encoding of $s_1 \dots s_n$. By Lochs' Theorem.

$$\frac{k}{n} \sim \frac{H(\mathcal{S})}{H(\frac{1}{d}, \dots, \frac{1}{d})} = \frac{h(\mathcal{S})}{\log_d d} = H(\mathcal{S}),$$

for the vast majority of the strings[6]. Thus we get compression by a factor $H(\mathcal{S})$, with maybe 2 bits excess to make sure that $D \subset J$, rather than taking the first k digits of some $x \in J$.

The algorithm describe here, can be performed using two maps on the unit interval, $T_d(x) = dx \pmod 1$ and

$$T_p(x) = \frac{1}{p_s}(x - \ell_s) \quad \text{if} \quad x \in J_s,$$

where $J_s := [\ell_s, r_s)$ is the interval of the first partition belonging to symbol s. That is, if we order the $s \in \mathcal{S}$, the $\ell_s = \sum_{s' < s} p_{s'}$ and $r_s = \ell_s + p_s$. Next, with input string $s_1 \dots s_n \in \mathcal{S}^n$ perform the following steps:

1. $x := 0$; $y := 1$;
2. For $i = 1$ to n do:
   $x := T_p^{-1}(x) \cap J_{s_i}$; $y := T_p^{-1}(y) \cap J_{s_i}$;

[5]Indeed, without mentioning Lochs.

[6]For Lebesgue almost every $x \in [0, 1]$, if $J \ni x$, then $k/n \sim H(\mathcal{S})$.

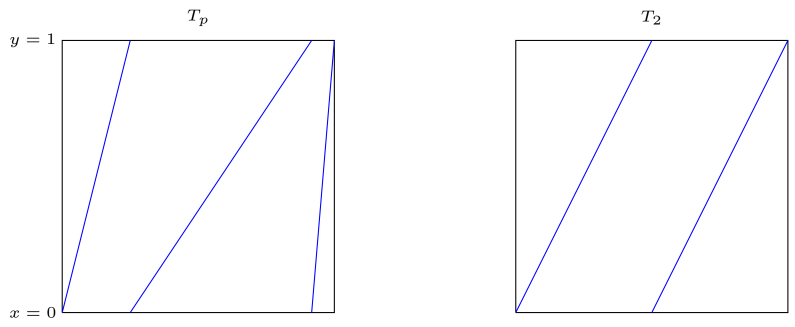


Figure 16: The maps T_p and T_d for $p = (\frac{1}{4}, \frac{2}{3}, \frac{1}{12})$ and $d = 2$.

3. If x and y are already expressed in base d, take the longest common prefix of the expansions of x and y as $c(s_1 \dots s_n)$
4. Otherwise continue as follows: $k := 1$;
5. While $\frac{i}{d} \notin [x, y]$ for every $i \in \{1, \dots, d-1\}$ do:
   $a_k := \lfloor dx \rfloor$; $k := k + 1$; $x := T_d(x)$; $y := T_d(y)$;
6. Set $c(s_1 \dots s_n) = a_1 \dots a_{k-1}$.

As with the Huffman coding, arithmetic coding is easily adapted to changing the probability vector $(p_s)_{s\in\mathcal{S}}$; we only have to adapt the map T_p in each step of the algorithm. Arithmetic coding is used in arithmetic codes, JBIG, dejavu and also an automatic writing tool called Dasher[7] designed for disabled people, among others.

2.5 ZLV Coding

The Lempel-Ziv-Welch [29] encoding algorithm (ZLV) was designed in 1984 after an earlier version by the first two authors [32, 33]. It encodes the source using a dictionary of labeled words, initially containing only the one-letter words. As the source is read, new labeled words are added to the dictionary which can have the structure of a binary (or d-ary if $\#\mathcal{A} = d$) tree for easy searching. The message is scanned left-to-right until a word is found that is not yet in the dictionary. This word is necessarily one letter, say a, longer than the best fit w in the dictionary so far, and the new word wa is put in the dictionary with the next available label. Instead of transmitting the whole dictionary, only the labels and the single "extra" letters are transmitted, that is, only the label of w and a are transmitted.

As an example, we use the Feigenbaum sequence as source and the initial dictionary if 0 and 1 with labels 0000and 0001 respectively. We use 4-bit labels, for longer messages longer (frequently 12-bit) labels are necessary. This leads to Tabel 2.

[7]See http://www.inference.org.uk/dasher/

ρ_{feig}	10	11	101	01	011	1011	10111	010	101110	1010
label	2	3	4	5	6	7	8	9	10	11
transmit	1 0	1 1	2 1	0 1	5 1	4 1	7 1	5 0	8 0	4 0
in binary	0001 0	0001 1	0010 1	0000 1	0101 1	0100 1	0111 1	0101 0	1000 1	0100 0

Table 2: Encoding the Feigenbaum sequence by ZLV.

We see that this 34 bit message is transmitted using 50 bits, but the true (and asymptotically optimal) compression kicks after some time. The compression is best if there are many repetitions in the message. In particular, sources of zero entropy work well: the smaller the word-complexity $p(n)$, the better.

For the decoding, the dictionary is build at the decoding end, again with initial dictionary 0 and 1 with labels 0000 and 0001 respectively, see Table 3.

code	0001 0	0001 1	0010 1	0000 1	0101 1	0100 1	0111 1	0101 0	1000 1	0100 0
label	2	3	4	5	6	7	8	9	10	11
label + letter	1 0	1 1	2 1	0 1	5 1	4 1	7 1	5 0	8 0	4 0
decoded	10	11	101	01	011	1011	10111	010	101110	1010

Table 3: Decoding the Feigenbaum sequence by ZLV.

ZLV encoding doesn't rely on a particular alphabet, letter frequencies or adapted letter frequencies. It was initially used in GIF-compression, but more frequently in other application (e.g. the UNIX compression tool, in gzip, PKZIP, and occasional for pdf-file compression in Acrobat reader) when the initial patents ran out.

References

[1] J.-P. Allouche, J. Shallit, *The ubiquitous Prouhet-Thue-Morse sequence*, Sequences and their applications, (Singapore, 1998), 1–16, Springer Ser. Discrete Math. Theor. Comput. Sci., Springer, London, 1999.

[2] J.-P. Allouche, J. Shallit, *Automatic sequences: theory, applications, generalizations*, Cambridge Univ. Press, 2nd. edition, Cambridge (2003).

[3] J. Berstel, A. Lauve, C. Reutenauer, F. Saliola, *Combinatorics on words. Christoffel words and repetitions in words*, CRM Monograph Series **27** (2009), Providence, RI: Amer. Math. Soc.

[4] J. Büchi, *Weak second-order arithmetic and finite automata*, Z. Math. Logik Grundlagen Math. **6** (1960), 66–92.

[5] A. Cobham, *Uniform tag sequences*, Math. Systems Theory **6** (1972): 164–192.

[6] F. Durand, *A generalization of Cobham's theorem*, Theory of Computing Sys. **31** (1998), 169–185.

[7] F. Durand, M. Rigo, *On Cobham's theorem*, Chapter in Automata: from Mathematics to Applications, Eur. Math. Soc., Editor J.-E. Pin,

[8] F. Durand, *Cobham's theorem for substitutions*, J. Eur. Math. Soc. **13** (2011), 1797–1812.

[9] R. Fano, *The transmission of information*, MIT Technical Report no. 65 (196).

[10] S. Golomb, *Run-length encodings*, IEEE Transactions on Information Theory, **12** (1966), 399–401.

[11] V. Goppa, *A new class of linear error correcting codes*, Probl. Peredach. Inform. **6** (1970), 24–30.

[12] V. Goppa, *Rational representation of codes and* (L, g) *codes*, Probl. Peredach. Inform. **7** (1971), 41–49.

[13] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addision-Wesley Publ. ISBN 0-201-02988-X

[14] D. Huffman, *A method for construction of minimum-redundancy codes*, Proc. of IRE **40** (1952), 1098–1101.

[15] T. Krebs, *A more reasonable proof of Cobham's theorem*, Preprint (2018) arXiv:1801.06704

[16] S. Ling, C. Xing, *Coding theory; a first course*, Cambridge Univ. Press, ISBN 978-0-521=52923-5, Cambridge, 2004.

[17] M. Lothaire, *Applied combinatorics on words*, Encyclopedia of Mathematics and Its Applications **105** (2005) 524–.

[18] D. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, 2003.

[19] M. Minsky, *Universality of* $(p = 2)$ *tag systems and a* 4 *symbol* 7 *state universal Turing machine*, 1962 Technical Report, Massachusetts Institute of Technology **201** Cambridge, MA.

[20] M. Morgenstern, *Turing machines with two letters and two states*, Complex Systems **19** (2010) 29–43.

[21] R. Pasco, —em Source coding algorithms for fast data compression, Ph.D. thesis, Dept. of Electrical Engineering, Stanford Univ., Stanford, Calif. (1976).

[22] J. Preskill, *Quantum information and computation*, Lecture Notes for Physics **229**, CreateSpace Independent Publishing Platform (2015), see also Chapter 10 on arXiv:1604.07450.

[23] R. Rice, *Some practical universal noiseless coding techniques*, Technical Report 79/22, Jet Propulsion Laboratory, 1979.

[24] J. Rissanen, *Generalized Kraft inequality and arithmetic coding*, IBM 1. Res. Dev. **20** (1976), 198–203. Another early exposition of the idea of arithmetic coding.

[25] J. Rissanen, *Arithmetic codings as number representations*, Acta Polytech. Stand. Math. **31** (1979), 44–51.

[26] C. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, (1948), 379–423 & 623–656.

[27] M. Tsfasman, S. Vlăduţ, Th. Zink, *Modular curves, Shimura curves, and Goppa codes, better than Varshamov-Gilbert bound*, Math. Nachr. **109** (1982), 21–28.

[28] A. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proc. London Math. Soc. **42** (1937), 230–65.

[29] T. Welch, *A technique for high-performance data compression*, Computer **17** (1984), 8–19.

[30] I. Witten, R. Neal, J. Cleary, J *Arithmetic Coding for Data Compression*, Communications of the ACM. **30** (1987), 520–540.

[31] S. Wolfram, *Cellular automata and complexity: collected papers*, Westview Press, Boulder Colorado (1994).

[32] J. Ziv, *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory, **23** (1977), 337–343.

[33] J. Ziv, A. Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Transactions on Information Theory **24** (1978), 530.