Information Theory
Degree in Data Science and Engineering
Lesson 4: Source coding

Jordi Quer, Josep Vidal

Mathematics Department, Signal Theory and Communications Department
{jordi.quer, josep.vidal}@upc.edu

2019/20 - Q1

Purpose of the chapter

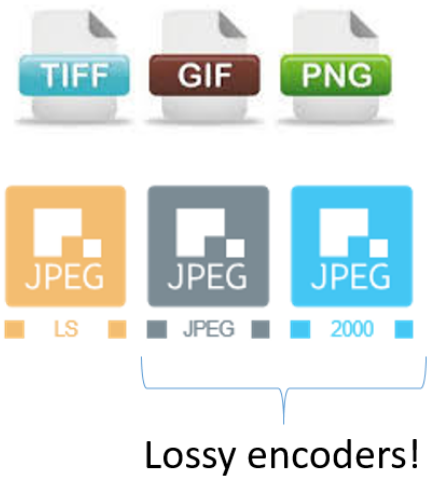
We proved in the previous chapter that a source can be encoded to a rate given by the entropy. That result holds for very large source words, assuming knowledge of the source statistics. We want to answer two questions:

- Can we build practical codes achieving a rate close to entropy for small values of n?
- Can we find affordable codes that compress to the entropy in the absence of knowledge of the source statistics?

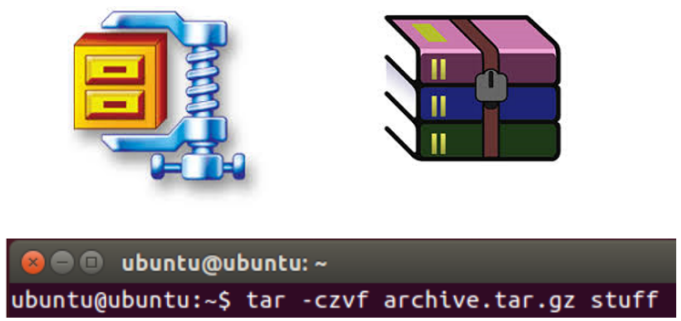
Popular source coders

Several source coders are being used in our daylife...

Lossless image compression standards



File compression software



Types of codes

- Symbol coding, on a per-symbol basis $\mathcal{C}_n : \mathcal{X}^n \to \mathcal{B}^*$
  These may be fixed length or *variable length*.
  Example: Code #1, $\mathcal{X} = \mathcal{B} = \{0, 1\}$, $n = 2$

$$\begin{array}{ccc} 00 & \to & 0 \\ 01 & \to & 01 \\ 10 & \to & 10 \\ 11 & \to & 101 \end{array}$$

- Stream codes, performed directly on streams $\mathcal{C}_* : \mathcal{X}^* \to \mathcal{B}^*$
  Example: Code #2

$$\begin{array}{ccc} 0 & \to & 000 \\ 1 & \to & 010 \\ 00 & \to & 110 \\ 01 & \to & 1110 \\ 10 & \to & 1111 \\ 11 & \to & 11011 \\ 000 & \to & 100010 \\ 001 & \to & 111011 \\ \ldots & \to & \ldots \end{array}$$

Properties

Some definitions for a code...

- Non-singularity: the coding has to be an injective map so that no two different source words are mapped onto the same codeword.
- Unique decodability: the extension code is not singular. Let us encode the following sequence with code #1:

$$001110101001011 \rightarrow \mathcal{C} \rightarrow \begin{array}{l} 0011010101010\ldots \\ 0011010101010\ldots \\ 0011010101010\ldots \\ 0011010101010\ldots \end{array}$$

How will decoding proceed? As the code is variable length a tree of possibilities opens up! To solve it, it is desirable that codes fulfill the prefix property...

- Prefix or instantaneous code: no codeword is prefix of another codeword.

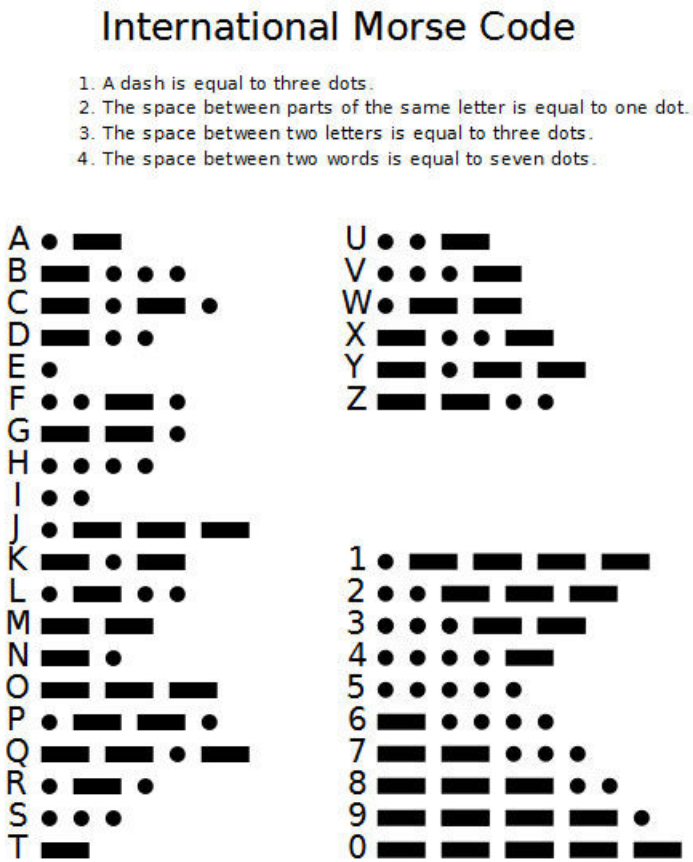
The Morse "code"

Considered as set of words over two-symbol alphabet consisting of two letters:

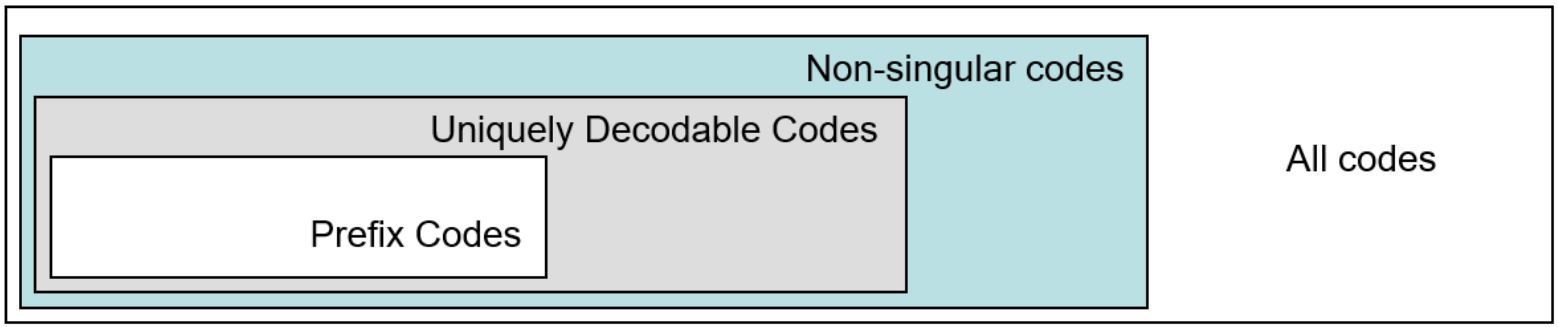
dot and dash,

the *Morse code* is not uniquely decodable.

Separation of letters and of words needs the use of spaces (of length 3 and 7).



Properties

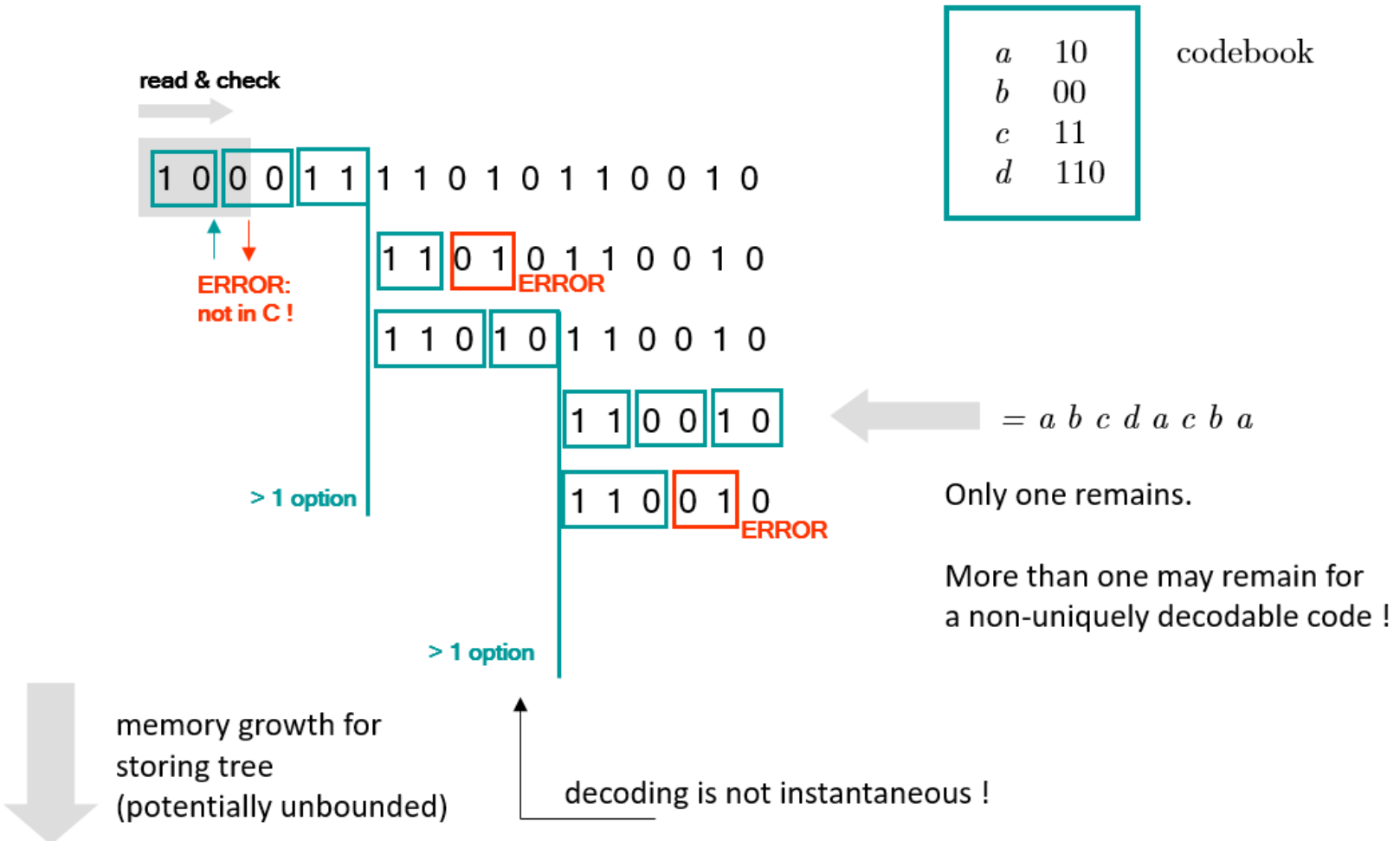


- **What non-singular codes are not uniquely decodable?** When one codeword can be made up of at least two codewords. For example:

U	singular	non-singular, not uniquely decodable	uniquely decodable, not instantaneous	instantaneous
a	0	0	10	0
b	0	010	00	10
c	0	01	11	110
d	0	10	110	111

A decoding procedure for a uniquely decodable non-instantaneous code... see over.

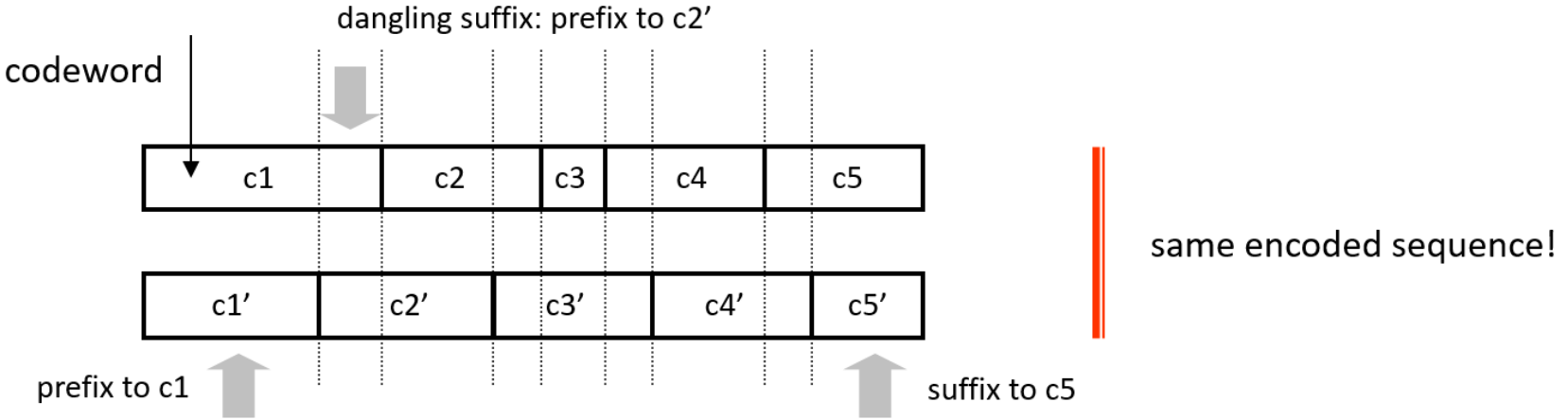
Properties



Decoding delay is incurred, so implementation costs increase. Does it pay to use a non-instantaneous code? The answer later on.

Unique decodability test

The *Sardinas-Patterson test* checks for unique decodability:



- Construct a set of all possible dangling suffixes.
- Unique decodability is attained iff the set contains no codeword.

- **What uniquely decodable codes are not instantaneous?** Those in which future symbols need to be examined before decoding.

Constraints for unique decodability

With the objective of minimizing the average number of bits per symbol, we can assign less bits to the more likely symbols. This necessarily entails lengthening other codewords to preserve unique decodability.

Symbol	Code #1	Probability	Code #2
a	000	0.5	0
b	001	0.25	1 0 0
c	010	0.05	1 0 1
d	011	0.05	1 1 0 0 0
e	100	0.05	1 1 0 0 1
f	101	0.05	1 1 0 1 0
g	110	0.04	1 1 0 1 1
h	111	0.01	1 1 1 0 0

Is it possible to formalize this constraint on the lengths of the codewords? Note that the prefix condition on codewords is enough to guarantee unique decodability...

Constraints for unique decodability

Let us assume an input alphabet $\mathcal{X}$ and a D-ary code

$$\mathcal{C} : \mathcal{X} \to \mathcal{D}^*$$

where $D = |\mathcal{D}|$. Let $c_j = \mathcal{C}(x_j)$, and $\ell_j = \ell(c_j)$ be the length of codeword c_j. For a **prefix code**, the lengths satisfy the following constraint:

Theorem (Kraft's inequality)

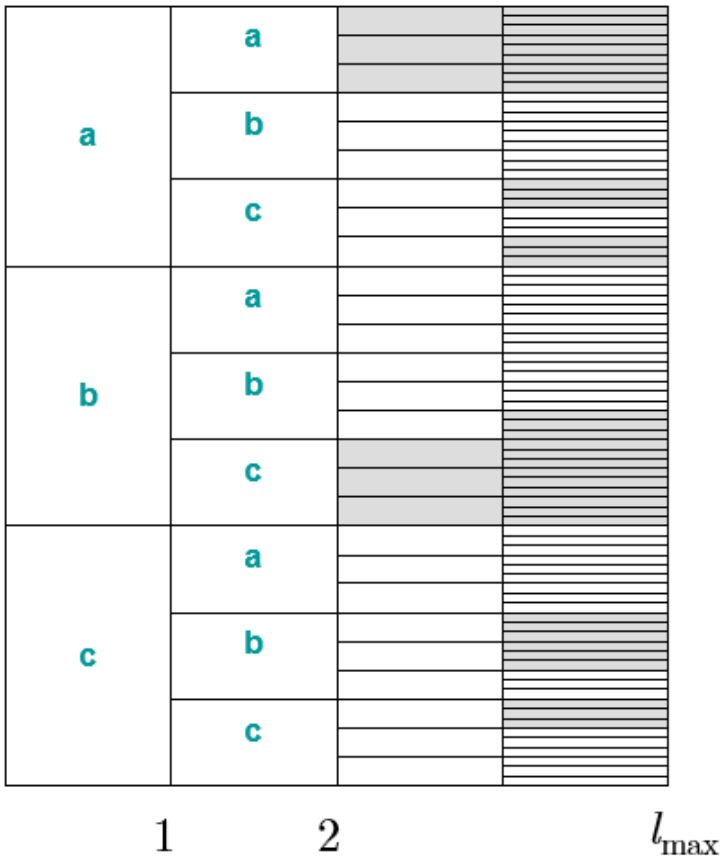
For any D-ary prefix code, the codeword lengths satisfy

$$\sum_i D^{-\ell_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

Proof. Take a D-ary tree in which each node has D children, the branches representing the symbols of the codeword. The prefix condition implies that no codeword is an ancestor of any other codeword in the tree: each codeword removes its descendants as possible codewords.

Constraints for unique decodability



Proof (cont.). Let ℓ_{max} be the length of the longest codeword. A codeword at level ℓ_i has $D^{\ell_{max}-\ell_i}$ descendants at level ℓ_{max}. Each of the descendant sets must be disjoint. Also the total number of nodes in these sets must be less than or equal to $D^{\ell_{max}}$.

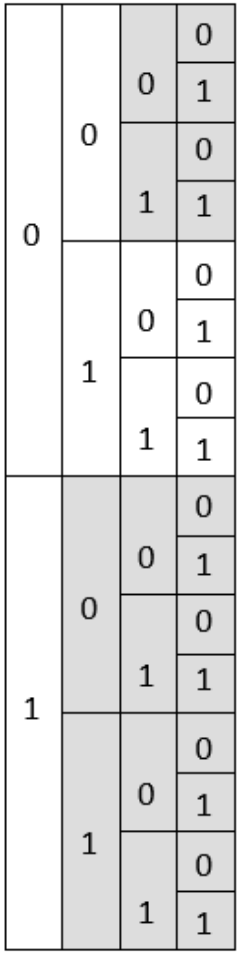
Hence summing over all codewords:

$$\sum_i D^{\ell_{max}-\ell_i} \leq D^{\ell_{max}}$$

The converse is true by construction of the codes. □

A code satisfying $\sum_i D^{-\ell_i} = 1$ is called *complete*.

Examples: codes for $D = 2$



a	00	(2)
b	0100	(4)
c	0101	(4)
d	0110	(4)
e	0111	(4)
f	1	(1)

codebook
(complete code)

$$\frac{1}{2^2} + \frac{4}{2^4} + \frac{1}{2^1} = 1 \leq 1$$

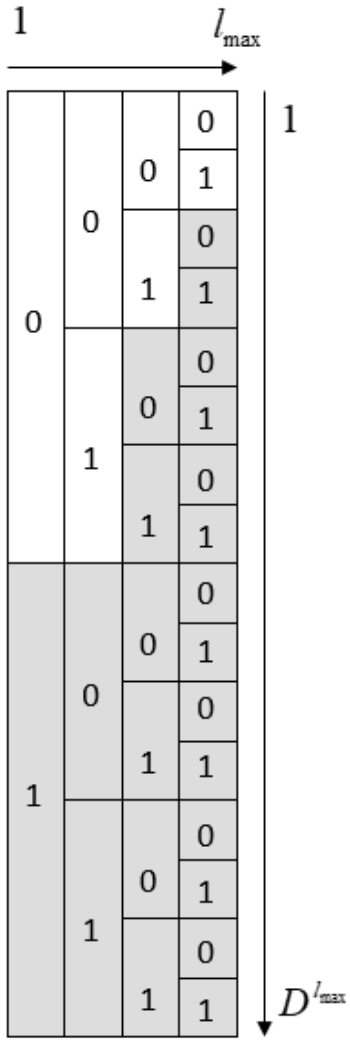
The right amount of bits ??

a	0000	(4)
b	0001	(4)
c	001	(3)
d	01	(2)

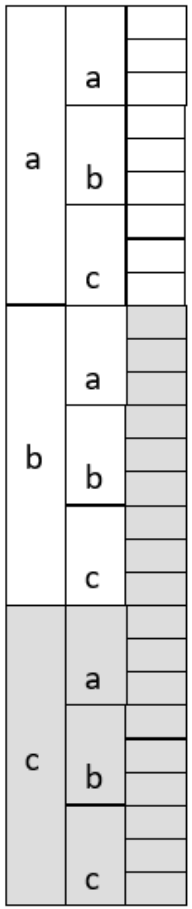
codebook
(non-complete code)

$$\frac{2}{2^4} + \frac{1}{2^3} + \frac{1}{2^2} = \frac{1}{2} \leq 1$$

Too many bits ??



Examples: codes for $D = 3$



s1	aaa	(3)	s7	aca	(3)
s2	aab	(3)	s8	acb	(3)
s3	aac	(3)	s9	acc	(3)
s4	aba	(3)	sa	ba	(2)
s5	abb	(3)	sb	bb	(2)
s6	abc	(3)	sc	bc	(2)

code #|
alphabet = 12
non-complete

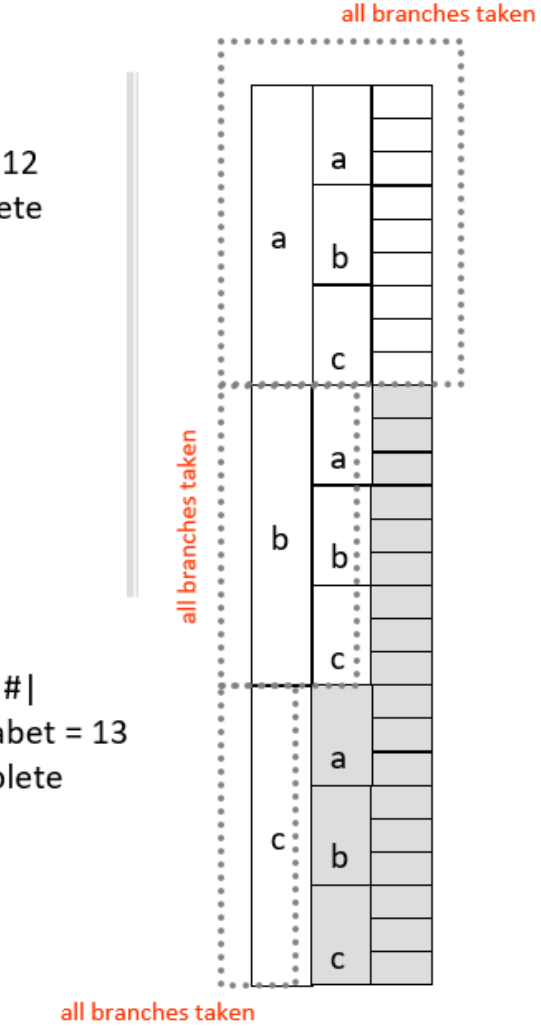
$$\frac{9}{3^3} + \frac{3}{3^2} = \frac{2\times 3^2}{3^3} \leq 1$$

s1	aaa	(3)	s7	aca	(3)
s2	aab	(3)	s8	acb	(3)
s3	aac	(3)	s9	acc	(3)
s4	aba	(3)	sa	ba	(2)
s5	abb	(3)	sb	bb	(2)
s6	abc	(3)	sc	bc	(2)
			sd	c	(1)

code #|
alphabet = 13
complete

$$\frac{9}{3^3} + \frac{3}{3^2} + \frac{1}{3^1} = 1 \leq 1$$

remove one at end and the code is not complete



McMillan inequality

The class of **uniquely decodable codes** is larger than the class of instantaneous codes, so one might expect to achieve a lower average codeword length if not being restricted to prefix codes. However...

Theorem (McMillan inequality)

The codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$, with $m = |\mathcal{X}^n|$, of a uniquely decodable D-ary code must satisfy the Kraft's inequality

$$LEF(D) = \sum_{i=1}^{m} D^{-\ell_i} \leq 1$$

Conversely, given a set of codeword lenghts satisfying this equality, there exists a uniquely decodable code with these codeword lenghts.

LEF is the acronim for *length enumeration function* of the code.

McMillan inequality

Proof. Assume a uniquely decodable encoding $\mathcal{C} : \mathcal{X} \to \mathcal{D}^*$. For an n-symbol sequence $x^n \in \mathcal{X}^n$, $\ell(\mathcal{C}(x^n)) = \sum_{i=1}^n \ell(\mathcal{C}(x_i))$.

Now, let us construct

$$\left(\sum_{x\in\mathcal{X}} D^{-\ell(\mathcal{C}(x))}\right)^n = \sum_{x_1\in\mathcal{X}}\sum_{x_2\in\mathcal{X}}\cdots\sum_{x_n\in\mathcal{X}} D^{-\ell(\mathcal{C}(x_1))}D^{-\ell(\mathcal{C}(x_2))}\ldots D^{-\ell(\mathcal{C}(x_n))}$$

$$= \sum_{x^n\in\mathcal{X}^n} D^{-\ell(\mathcal{C}(x^n))} = \sum_{i=1}^{n\ell_{max}} a(i)D^{-i}$$

where $a(m)$ is the number of source sequences x^n mapping into codewords of length m.

The following facts will be used now:
- Since the code is uniquely decodable, at most one sequence x^n is mapped into a codeword.
- There are at most D^m code sequences of lengh m.

McMillan inequality

Proof (cont). Thus, $a(m) \leq D^m$, so we have

$$\left(\sum_{x \in \mathcal{X}} D^{-\ell(\mathcal{C}(x))} \right)^n = \sum_{i=1}^{n\ell_{max}} a(i) D^{-i} \leq \sum_{i=1}^{n\ell_{max}} D^i D^{-i} = n\ell_{max}$$

This has to be valid for all n. While the left term grows exponentially with n, the right term only linearly, so the only possibility for the inequality to hold is

$$\sum_{x \in \mathcal{X}} D^{-\ell(\mathcal{C}(x))} \leq 1$$

The converse is true by the converse of the Kraft inequality theorem and the fact that any prefix code belongs to the class of uniquely decodable codes. □

Implications of the Kraft-McMillan theorem

- No set of codeword lengths exists for a uniquely decodable code better than those associated with a prefix code.
- Hence, no advantage is gained from using uniquely decodable non-prefix codes.
- If $LEF(D) > 1$, the code is non-uniquely decodable (but not conversely, non-uniquely decodable codes with $LEF(D) \leq 1$ do exist).

Optimization of the average code length

Assume that input symbols $\mathcal{X}$ have a probability given by p_i. We formulate the codebook construction as an optimization problem over an integer grid:

$$\underset{\ell_i}{\text{minimize}} \sum_i p_i \ell_i$$

$$\text{subject to } \sum_i D^{-\ell_i} \leq 1$$

We **relax** the problem: assume real non-negative values for ℓ_i and complete code (Kraft inequality is fulfilled with equality). Then use Lagrange multipliers for the constrained minimization:

$$J = \sum_i p_i \ell_i - \lambda \left(\sum_i D^{-\ell_i} - 1 \right)$$

Derive and equate to zero $\frac{\partial}{\partial \ell_i} J = p_i - \lambda D^{-\ell_i} \ln D = 0$ to obtain the lengths of a *Shannon code*:

$$\ell_{i,opt} = \log_D \frac{1}{p_i}$$

Optimization of the average code length

The average length of the code is

$$L_{opt} = \mathbb{E}\left[\ell_{opt}\right] = \sum_i p_i \ell_{i,opt} = -\sum_i p_i \log_D p_i = H_D(X)$$

equal to the entropy! However, since ℓ_i must be an integer we are forced to round the optimal values.

Theorem (Sandwich bound)

The average number of bits per codeword for a prefix code satisfies

$$H_D(X) \leq L < H_D(X) + 1$$

Proof. For the **lower bound**:

$$\begin{aligned} L - H_D(X) &= \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \\ &= -\sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \end{aligned}$$

Optimization of the average code length

Proof (cont). Define the auxiliary distribution $r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}}$ and define $d = \sum_i D^{-\ell_i} \leq 1$ so that

$$\begin{aligned} L - H_D(X) &= \sum_i p_i \log_D \frac{p_i}{r_i} - \log_D d \\ &= D(p||r) + \log_D d^{-1} \geq 0 \end{aligned}$$

where the Kullback-Leibler divergence and the last term are positive.

For the **upper bound**, we take the following word lengths: $\ell_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil < \log_D \frac{1}{p_i} + 1$, and then

$$L = \sum_i p_i \ell_i \leq \sum_i p_i \log_D \frac{1}{p_i} + \sum_i p_i < H_D(X) + 1 \qquad \square$$

Tightness of the bounds

The extra 1 can be critical for small length codes. What can be done?

Encode n-symbols blocks:

$$nH_D(X) = H_D(X^n) \leq L_n < H_D(X^n) + 1$$

$$\frac{1}{n}H_D(X^n) \leq \frac{1}{n}L_n < \frac{1}{n}H_D(X^n) + \frac{1}{n}$$

so that when $n \to \infty$, $L = \frac{1}{n}L_n \to H(X)$

Which factors affect the tightness of the Sandwich bound?

Tightness of the bounds

...for the **lower bound**, in the equation

$$L - H_D(X) = D(p||r) + \log_D d^{-1} \geq 0$$

the first term is due to the mismatch of density functions due to quantisation of probabilities, and the second in the excess in codewords (is zero if the code is complete).

The following properties hold:

- if $p = r$ the code is complete, but
- if the code is complete, not necessarily $p = r$.

As a consequence, the search for a code that minimizes the gap must involve the joint minimization of the two terms.

Tightness of the bounds

...for the **upper bound** we can write

$$\ell_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil = \log_D \frac{1}{p_i} + \epsilon_i, \text{ with } 0 \leq \epsilon_i < 1$$

When encoding a long sequence of symbols, the quantization error ϵ is amplified by the length of the sequence. Again, this can be solved by encoding blocks of n-symbols.

Example. Shannon code (that take the upper integer of optimum lengths) is not optimum when used in symbol codes. Assume a two symbol alphabet with probabilities $\{1/2^7, 1 - 1/2^7\}$. Although the minimum length codebook is $\{1, 0\}$, the Shannon codebook associated to those probabilities is $\{1111111, 0\}$ (note that it is an instantaneous code).

Shannon-Fano codes

How can we build prefix codes achieving the minimum average length given by the Sandwich bound?

The first attempt to construct optimal codes was in 1949 by Shannon and Fano. The Shannon-Fano codes are constructed in the following way:

- Sort the alphabet letters in decreasing order of probability $p_1 \geq p_2 \geq \cdots \geq p_r \geq p_{r+1} \geq \cdots \geq p_m$
- Divide them into two subsets with total probabilities as close as possible
- The first symbol of the codewords is assigned: those in the first subset get a 0, those in the second get a 1
- Divide again each of the two sets into two subsets with probabilities as close as possible, and choose the second symbol of the codewords accordingly
- Proceed in this way until arriving to sets with a single letter.

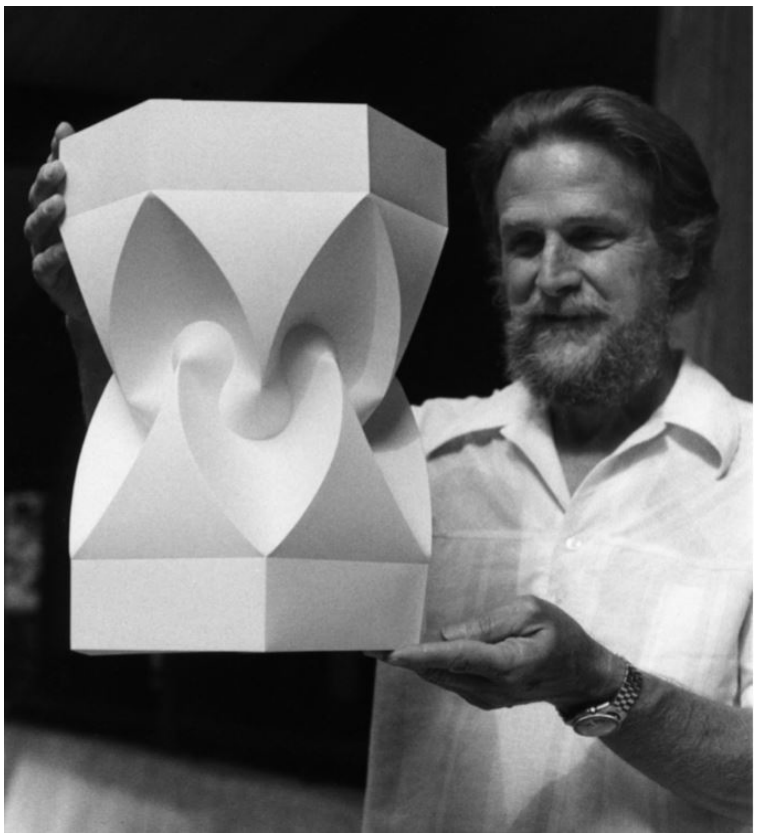
Shannon-Fano code

Here is an example of a Shannon-Fano code:

<table>
<tr><th>a_i</th><th>$p(a_i)$</th><th>1</th><th>2</th><th>3</th><th>4</th><th>Code</th></tr>
<tr><td>a_1</td><td>0.36</td><td rowspan="2">0</td><td colspan="3">00</td><td>00</td></tr>
<tr><td>a_2</td><td>0.18</td><td colspan="3">01</td><td>01</td></tr>
<tr><td>a_3</td><td>0.18</td><td rowspan="4">1</td><td colspan="3">10</td><td>10</td></tr>
<tr><td>a_4</td><td>0.12</td><td rowspan="3">11</td><td colspan="2">110</td><td>110</td></tr>
<tr><td>a_5</td><td>0.09</td><td rowspan="2">111</td><td>1110</td><td>1110</td></tr>
<tr><td>a_6</td><td>0.07</td><td>1111</td><td>1111</td></tr>
</table>

Shannon-Fano codes are “suboptimal”: they have very small average length but not always the smallest possible one, which is by definition the average length of the optimal codes.

Huffman codes



David Huffman (1925-1999)

In 1951 Prof. *Robert Fano* was teaching a course on Information Theory at MIT. He proposed the students to work on a term paper about optimal source codes instead of going to the final exam.

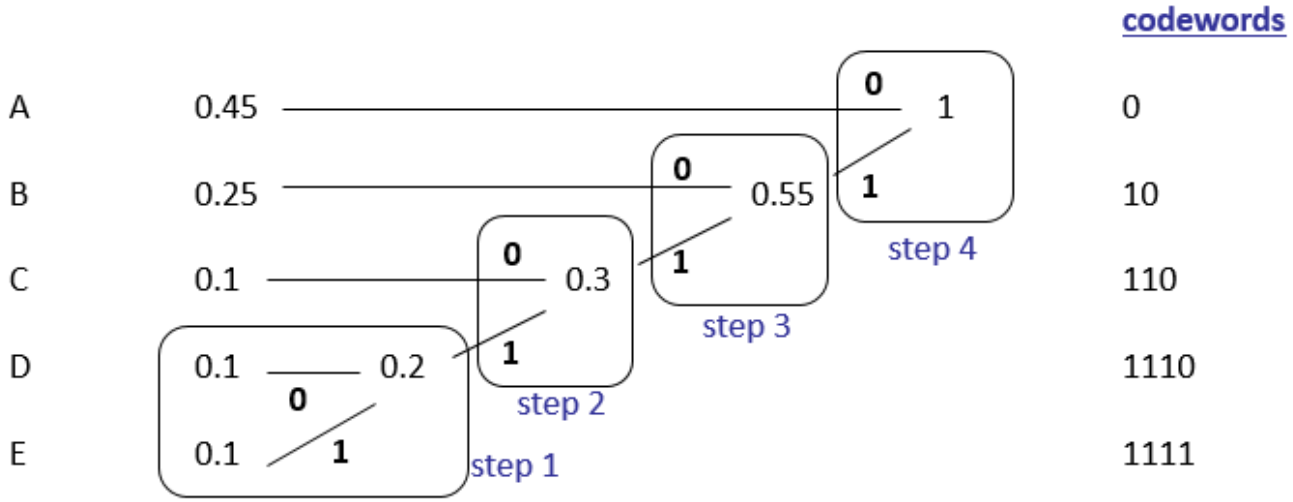
The paper by student *David Huffman* proposed a prefix code that was shown to be optimal (shortest average length).

Huffman codes are used in many compression standards today.

Let us introduce the Huffman code construction through examples before proving optimality.

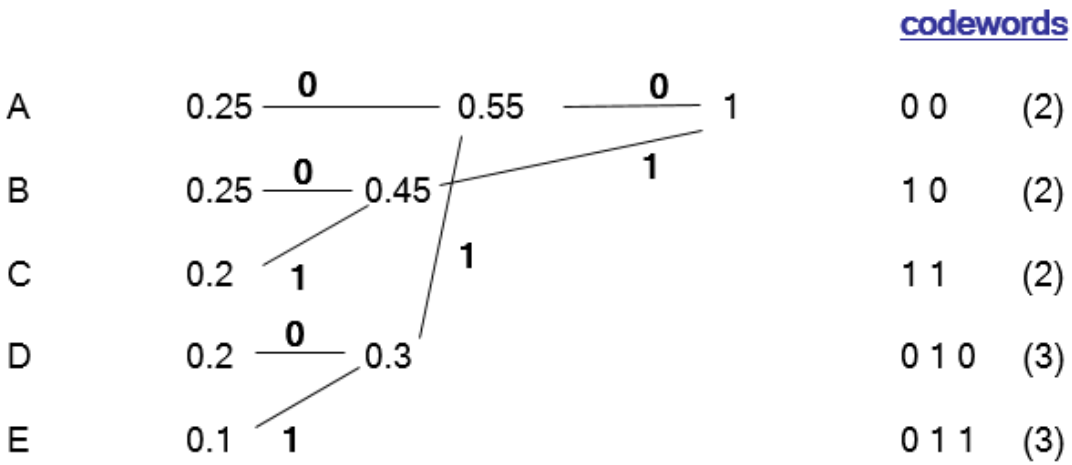
Example 1

Let us illustrate with an example where $|\mathcal{X}| = 5$ and $D = 2$...



- Order all symbols by decreasing probability.
- Group the two least likely nodes into one node and add probabilities. The number of nodes will decrease by one at each step.
- Label the upper and lower branches of node merging 0 and 1.
- Proceed so until one node is left with unit cumulative probability.
- Codewords are generated by reading backwards from the tree root.

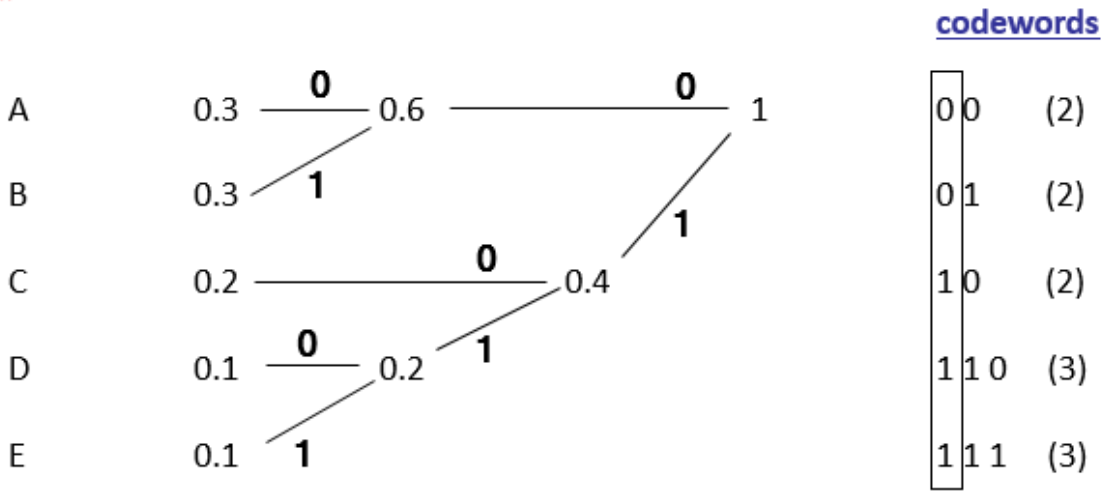
Example 2



$$LEF(2) = \frac{3}{2^2} + \frac{2}{2^3} = 1$$

Example 3

Does the Huffman procedure always generate a complete code?

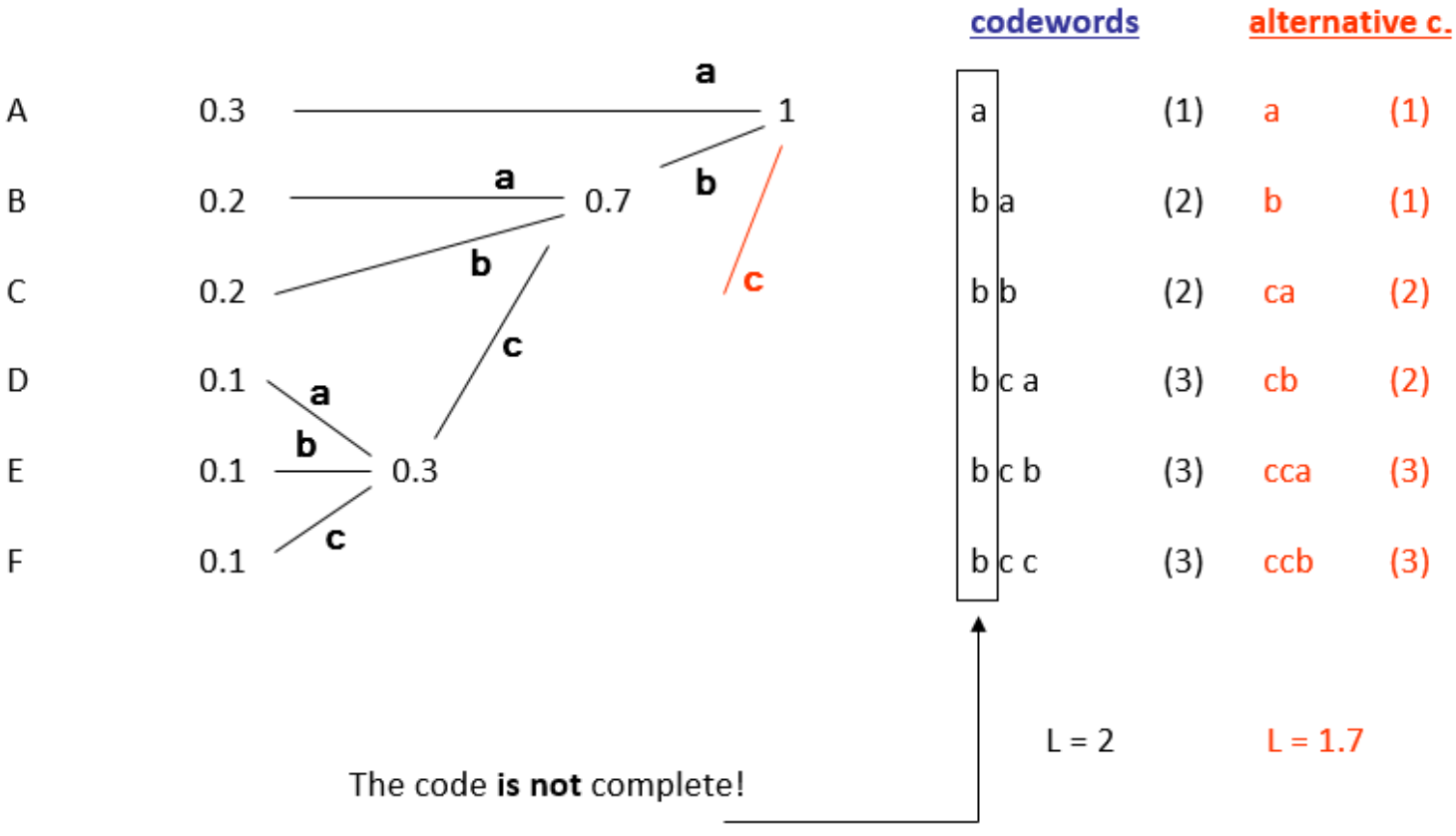


$$LEF(2) = \frac{3}{2^2} + \frac{2}{2^3} = 1$$

The code is always complete in the binary case!

Example 4

In the non-binary case, it is not always complete. One case for $D = 3$...



$$LEF(3) = \frac{2}{3} + \frac{2}{3^2} + \frac{2}{3^3} = \frac{26}{27}$$

Should the procedure be modified?

Huffman procedure for $D > 2$

If $D \geq 3$, we may not have sufficient number of symbols to combine them D at a time.

In such case we add the mininum number of dummy symbols of 0 probability at the end of the tree, so that the total number of symbols is $1 + K(D-1)$ where K is the total number of merges.

Add the required number of dummy symbols in example 4!

Check that the code is complete if we include the dummy symbols in the evaluation of the LEF.

Example: short text source

Consider the string of text:

`setzejutgesdunjutgatmengenfetgedunpenjat`

It contains 40 letters. Counting the appearances we obtain:

LETTER	a	d	e	f	h	j	m	n	p	s	t	u	z
count	2	2	8	1	3	4	1	5	1	2	6	4	1

Let us generate a source X with 13 letters and its probability of appearance. The entropy is

$$H(X) = 3.375071\ldots, \qquad \log_2(13) = 3.700440\ldots$$

The next table contains the code in the proof of Shannon source code theorem, the Shannon-Fano code and the Huffman code for this source, with the corresponding average lengths:

Example: short text source

LETTER	Prob.	Shannon	Shannon-Fano	Huffman
a	0.05	01100	11100	11100
d	0.05	11100	1101	11101
e	0.2	000	00	00
f	0.025	001100	111111	111100
h	0.075	1100	1011	1011
j	0.1	0010	1010	010
m	0.025	101100	111110	111101
n	0.125	100	011	100
p	0.025	011100	11110	111110
s	0.05	11010	1100	1010
t	0.15	010	010	110
u	0.1	1010	100	011
z	0.025	111100	11101	111111
Average length	3.375	3.875	3.425	3.425

Coincidentaly, same length for Shannon-Fano and for Huffman.

Exercise

Determine the Huffman codes of the following two symbol sources, for $D = 4$ and for $D = 2$ in each case, and check for completeness.

A	0.3
B	0.2
C	0.2
D	0.1
E	0.1
F	0.1

A	0.2
B	0.2
C	0.2
D	0.1
E	0.1
F	0.1
G	0.1

a) Compare the entropy of the source with the average length of the encoded sequences. Check the LEF and relate completeness and average length.

b) Check that the probability distribution of symbols on the encoded sequence is closer to uniform than that of source symbols. Why?

c) Take one of the sources and build the Huffman code for pairs of input symbols (assumed independent) for $D = 2$. Solve the previous question again and compare the efficiency of the code.

Optimality of codes

For any distribution, there exist an optimal prefix code (with minimum expected length) $\mathcal{C}$ with $D = 2$ that satisfies the following properties:

Proposition (1)

If $p_i > p_j \;\rightarrow\; \ell_i \leq \ell_j$

Proposition (2)

The two longest codewords have the same length.

Proposition (3)

The two longest codewords, corresponding to the two least likely symbols, differ in the last binary digit.

Proof. P1 We consider a new code $\mathcal{C}'$ by swapping the assignment of symbols to codewords in $\mathcal{C}$

$$L_{\mathcal{C}'} - L_{\mathcal{C}} = \sum_k p_k(\ell'_k - \ell_k) = p_j\ell_i + p_i\ell_j - p_j\ell_j - p_i\ell_i = (p_j - p_i)(\ell_i - \ell_j)$$

since $\mathcal{C}$ is optimal the difference must be positive and P1 holds.

Optimality of codes

Proof (cont.)

P2. If the two longest codewords of $\mathcal{C}$ are not of the same length, the last binary digits of one can be deleted (since code is prefix) thus obtaining a lower average length, which is a contradiction!

P3. From P1, the longest codewords c_{m-1} and c_m are associated with the least likely symbols and by P2 they are of the same length. Assume we remove the last binary digit of the last one $c_m = c'||a$. Then $c_1, c_2, \ldots, c_{m-1}, c'$ is a shorter code and hence cannot be prefix. Therefore c' must be prefix of c_{m-1}. Hence c_{m-1} and c_m have the same prefix and only differ in the last symbol. $\square$

Optimality of codes

For the **non-binary case** ($D > 2$) the last propositions should be changed to

Proposition (2)

There exist at least two longest codewords with the same length.

Proposition (3)

The longest codewords differ in the last symbol.

Optimality of Huffman codes

Theorem

Huffman codes are prefix codes of minimum average length.

Proof. Assume an m-symbol input alphabet, a code $\mathcal{C}_m$ and proceed by induction. Let us prove it for $D = 2$.

- Take the longest codewords, by construction they differ in the last symbol so P3 holds.
- We construct a merged code of $(m-1)$ symbols where
  a) the common prefix of the two original longest codewords becomes the new codeword for the new merged symbol of length ℓ'_{m-1}.
  b) the probability of the merged symbol is the addition of the original probabilities.

Optimality of Huffman codes

Proof (cont.)

- The average length of the code is

$$\begin{aligned} L_{\mathcal{C}_m} &= \sum_{i=1}^{m} p_i \ell_i = \sum_{i=1}^{m-2} p_i \ell_i + (\ell'_{m-1} + 1) p_{m-1} + (\ell'_{m-1} + 1) p_m \\ &= \sum_{i=1}^{m-2} p_i \ell_i + \ell'_{m-1}(p_{m-1} + p_m) + (p_{m-1} + p_m) \\ &= L_{\mathcal{C}_{m-1}} + (p_{m-1} + p_m) \end{aligned}$$

- Now, the average length of the code $\mathcal{C}_m$ can be minimized by minimizing the length of the merged code $\mathcal{C}_{m-1}$, which should also have the structure of an optimum code.
- Proceed recursively! □

Note that the minimum average length property does not imply achieving the entropy. Only if the conditions given by the Sandwich bound hold.

Example: Huffman code of English source

LETTER	#	CODE	LETTER	#	CODE	LETTER	#	CODE
	216026	111	i	65471	0110	r	52173	0001
a	77948	1010	j	1084	1011111010	s	64256	0101
b	16886	100101	k	8058	10111111	t	88037	1101
c	22523	110011	l	42804	11000	u	26707	00001
d	38233	10110	m	23285	00000	v	8606	1011110
e	117141	001	n	65645	0111	w	22227	110010
f	20846	101110	o	69357	1000	x	1034	1011111001
g	20827	100111	p	17265	100110	y	16877	100100
h	62917	0100	q	1556	10111111011	z	632	1011111000

Source: Moby Dick, by Herman Melville.
1 168 421 characters of an alphabet of 27 letters.
$\log_2(27) = 4.75489\ldots$

Entropy: $H(X) = 4.094353\ldots$ bits.

Average length of code $\mathbb{E}[\ell(c(x))] = 4.134070\ldots$ bits per character.

Example: Huffman code of Spanish source

LETTER	#	CODE	LETTER	#	CODE	LETTER	#	CODE
	381208	00	i	90077	11101	s	125728	1010
a	200499	010	j	10530	11110100	t	61749	10011
b	24147	1011111	l	89143	11100	u	79560	11010
c	59437	10010	m	44658	111100	v	17856	1011011
d	87240	11011	n	112683	1000	w	2	11110101000
e	229191	011	o	162514	1100	x	377	11110101001
f	7581	111101011	p	35465	100110	y	25115	1111011
g	17225	1011010	q	32483	101100	z	6491	1111010101
h	19920	1011110	r	100955	11111			

Source: Don Quijote de la Mancha, by Miguel de Cervantes.
2 021 834 characters of an alphabet of 26 letters.
$\log_2(26) = 4.70044\ldots$

Entropy: $H(X) = 3.968888\ldots$ bits.

Average length of code $\mathbb{E}[\ell(c(x))] = 4.017647\ldots$ bits per character.

Disadvantages of Huffman codes

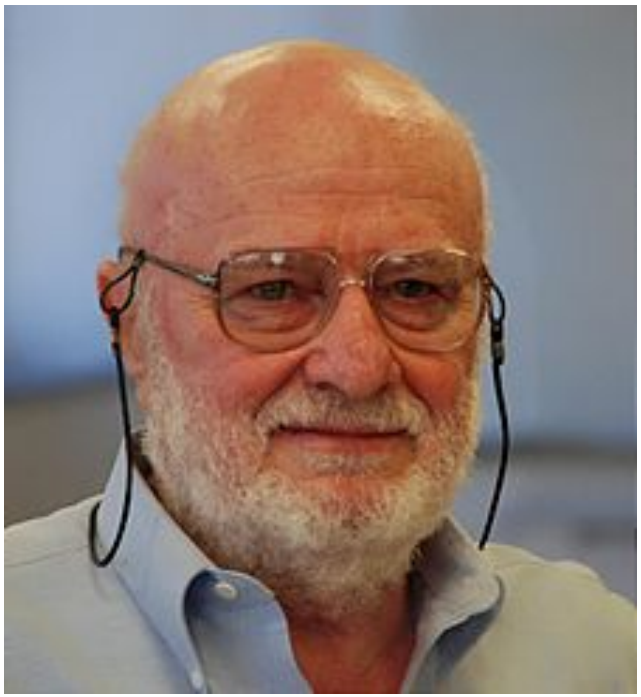
Huffman coding requires knowledge of the alphabet probabilities. For the case of unknown alphabet probabilities, adaptive versions have been developed in the literature.

A taxonomy of source codes...

- Huffman, Shannon-Fano-Elias $\rightarrow$ Distribution tailored
- Arithmetic $\rightarrow$ Probability models
- Lempel-Ziv $\rightarrow$ Universal, no probability models

Lempel and Ziv

In 1977 and 1978 *Abraham Lempel* and *Jacob Ziv*, engineers at Technion - Israel Institute of Technology, publish two papers introducing new ideas that will revolutionize lossless compression.



Abraham Lempel (1936–)

Jacob Ziv (1931–)

Lempel and Ziv

The methods proposed by Lempel and Ziv in their papers are known by the acronyms *LZ77 and LZ78*.

Since then, dozens of similar methods, variations and improvements have been proposed, many of them protected by patents.

This led to a new family of methods that are known under the generic name of *dictionary methods*, which are the preferred choice for lossless compression in computer storage of data.

Many are named by adding other initials to Lempel and Ziv's: *LZSS* (Storer-Szymanski), LZK (Phil Katz's *deflate*), *LZW* (Welch), *LZMA* (Markov), etc.

Used by most popular compressors ZIP, PKZIP, *compress*, 7-zip, GIF, PNG etc.

LZ codes

The *Lempel-Ziv algorithms* are universal compression methods (for ergodic sources) that do not use any probabilistic model for the source. In a way, it "learns" the source distribution by keeping a logbook of previously generated sub-sequences within the same realization.

An **LZ parsing** of a sequence is the ordered collection of phrases (or sub-sequences) that have not appeared so far.

An incoming sequence of symbols is separated by commas into phrases, such that after each comma we examine incoming symbols after finding the shortest phrase not marked off before:

1 1 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 1 . . .

1, |1 0, |1 1, |0, |0 0, |1 0 1, |0 0 1, |1 1 1, |1 0 0, |0 1, |. . .

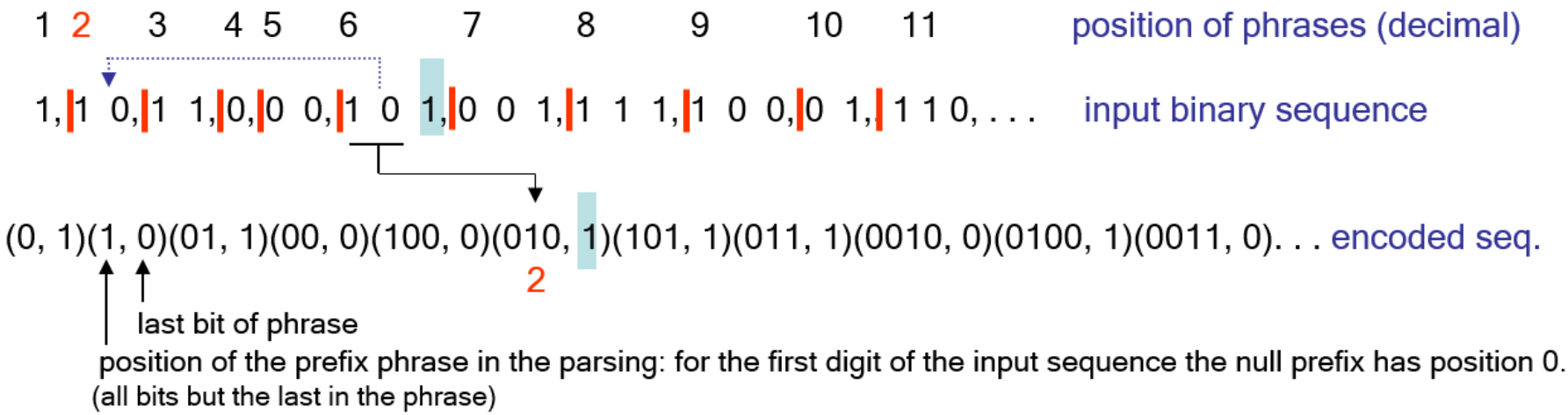
The LZ parsing is always distinct: no two phrases are identical.

LZ codes

Several versions of the LZ algorithm exist, we describe the philosophy of LZ78:

Encoder

- **Step 1**. A distinct parsing of the sequence x^n is generated by comparing the received sequence with those appeared so far that are stocked in a dictionary.
- **Step 2**. Each substring is encoded by using a prefix equal to the pointer to a substring that appeared earlier, and appending the new symbol:



- **Step 3**. The dictionary is updated at every received phrase as follows:

LZ codes

pointer	source	prefix size	encoded (prefix,bit)
1	1	1	0,1
2	10	1	1,0
3	11	2	01,1
4	0	2	00,0
5	00	3	100,0
6	101	3	010,1
7	001	3	101,1
8	111	3	011,1
9	100	4	0010,0
10	01	4	0100,1
11	110	4	0011,0

The prefix size is selected according to the maximum possible value of the prefix for each substring. This allows parsing the received sequence. Note that all codewords are different.

LZ codes

Decoder It is not needed to transmit the dictionary together with the data, as the decoder can build up the dictionary in the same way as the encoder does.

- **Step 1**. Identify the codeword in the received sequence by the knowing the prefix size.
- **Step 2**. Fill the receiver dictionnary row just like encoder does.
- **Step 3**. Decode the phrase by identifying the prefix in the dictionary.

LZ codes

Limitations

- The dictionary grows without bound.

- The gain of LZ encoding is observed for a sufficiently long sequence. For short sequences the encoding may be longer than the original sequence (as it happens in the example above).

- LZ is bad at capturing long range correlations, e.g. in two dimensional images of width w represented by sequences of pixels row-by-row. Similar pixels are w symbols apart.

Improvements

- Fix the dictionary once it is full, or throw it away and create a new one from scratch once a certain number of entries has been reached.

- Note that a shorter encoding sequence is possible: when the prefix has appeared twice it can be removed from the dictionary, shuffling indexes all along one and thus reducing the length of subsequent prefixes.

- The second time a prefix is used, we can be sure of the identity of the next bit so it is pointless to transmit it.

LZ codes

Lemma

Let $c(n)$ denote the number of phrases in the LZ parsing of an n-symbol binary sequence. Then

$$c(n) \leq \frac{n}{(1-\epsilon_n)\log n} \text{ where } \epsilon_n \to 0 \text{ as } n \to \infty$$

Theorem

For a stationary ergodic process of entropy rate $H(\mathbf{X})$ and $c(n)$ phrases in a distinct parsing, the length of the encoded sequences for large n is

$$\ell(c(x^n)) = c(n)(\log c(n) + 1)$$

and

$$\limsup_{n\to\infty} \frac{\ell(c(x^n))}{n} \leq H(\mathbf{X})$$

with probability 1.

Proof. Beyond scope, in T. Cover, *Elements of Information Theory*, chapter 13.

Example: First chapter of Quijote

Consider the string x^n of length $n = 10365$:

$$x^n = \text{“En un lugar de la Mancha ... había puesto.”}$$

The alphabet $\mathcal{X}$ contains 63 characters:

- 26 lowercase (including ñ) and 20 uppercase letters;
- 5 accented letters;
- 10 punctuation marks;
- 2 formatting characters: space and end of paragraph.

The most frequent are space (1867 times), e (1067), a (1017), o (734), s (571), etc. The less frequent are S (3 times), x, H, I, H (2) and O, U, ü, ¿, ?, ¡, ! (only once).

The corresponding source has entropy $H(X) = 4.27776\ldots$
Huffman code has average length $L = 4.31925\ldots$

Example: First chapter of Quijote

Codes comparison table

Compression method	Bits per symbol
ASCII code	8
Optimal block code	6
Shannon code	4.8383 ...
Shannon-Fano code	4.3234 ...
Huffman code	4.3192 ...
Arithmetic coding (= entropy)	4.2778 ...
LZ77	4.5422 ...
LZSS	4.1506 ...
LZK (deflate)	3.7269 ...
LZ78	4.4793 ...
LZW	3.9643 ...
Standard compression software	~ 3.5

Why the achieved bits per symbol in the compressed file is lower than $H(X)$?