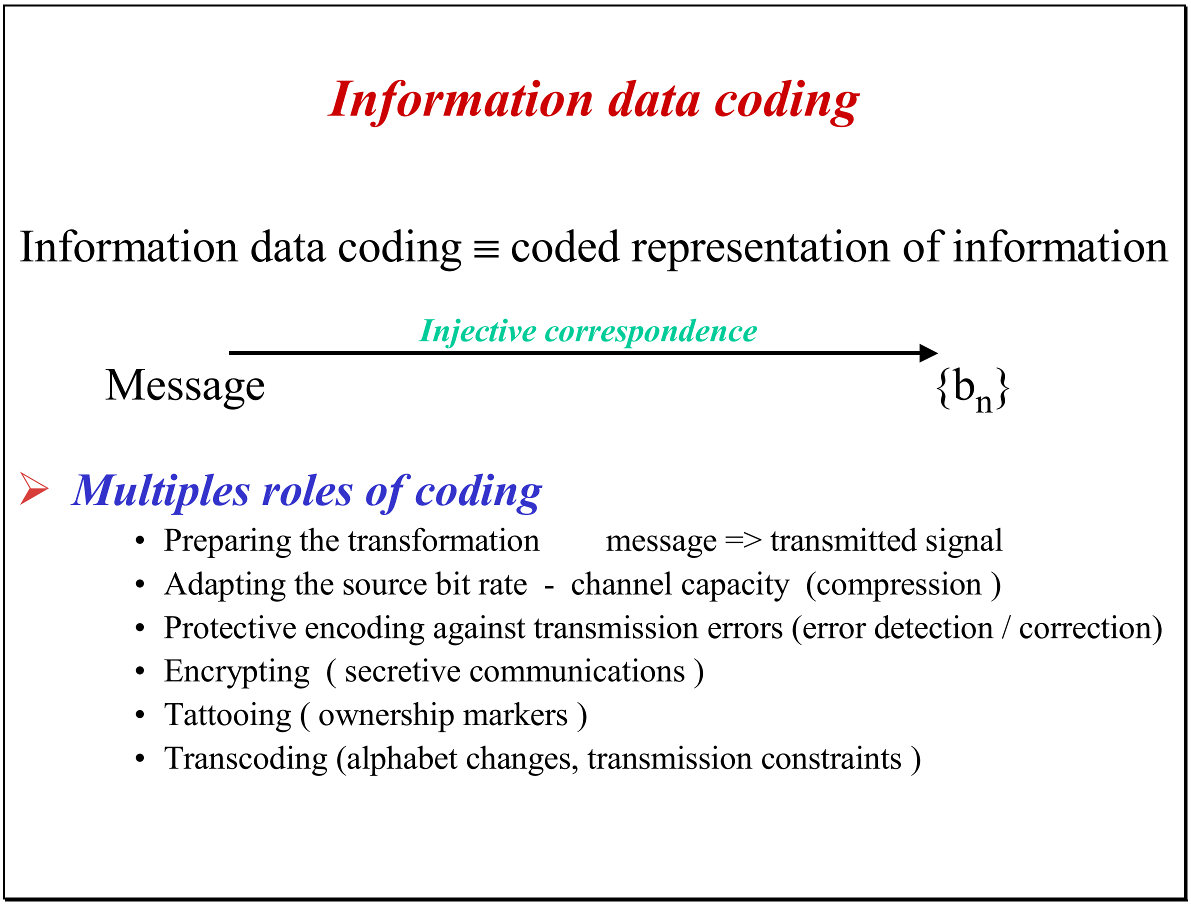
Chapter 5

STATISTICAL CODING

Presentation

Entropy



Information data coding

Information data coding ≡ coded representation of information

$$\text{Message} \xrightarrow{\textit{Injective correspondence}} \{b_n\}$$

➢ *Multiples roles of coding*

- Preparing the transformation message => transmitted signal
- Adapting the source bit rate - channel capacity (compression)
- Protective encoding against transmission errors (error detection / correction)
- Encrypting (secretive communications)
- Tattooing (ownership markers)
- Transcoding (alphabet changes, transmission constraints)

The goal of a communication system is to transport messages from a sender (the information source) towards a recipient (information user). The signal supporting the information being transmitted has to be compatible with the characteristics of the transmission channel. Information coding must establish an injective correspondence between the message produced by the source and the sequence of information $\{b_n\}$ sent to the transmitter.
This information coding plays numerous roles:

- It prepares the transformation of a message into a signal (carried out in the transmitter part: signal information encoding);
- It adapts the source of information to the capacity of the transmitting channel;
- It can be used to protect information against transmission errors (within certain limits), so as to detect and/or correct them (due to channel disturbances);
- It can also be used in certain cases for secret communications (encrypting) and watermarking to protect ownership.

A given transmission channel must be able to transport messages of various types that is why transcoding is necessary: this transforms the message representation from a codebook M_k using an alphabet A_k into a representation of the same message from a codebook M_0 using alphabet A_0. This particular alphabet is often the binary set $\{0, 1\}$, but this is not the only one.

Information data coding

➢ *Definitions*

- Message sources S: production of a sequence of messages, each of them being selected in a set M of messages
  (M : codebook of possible messages M = { m_1 , m_2 , ….}, the m_i are also called "words")

- Message: finite sequence of symbols
  (characters taken from A : alphabet)

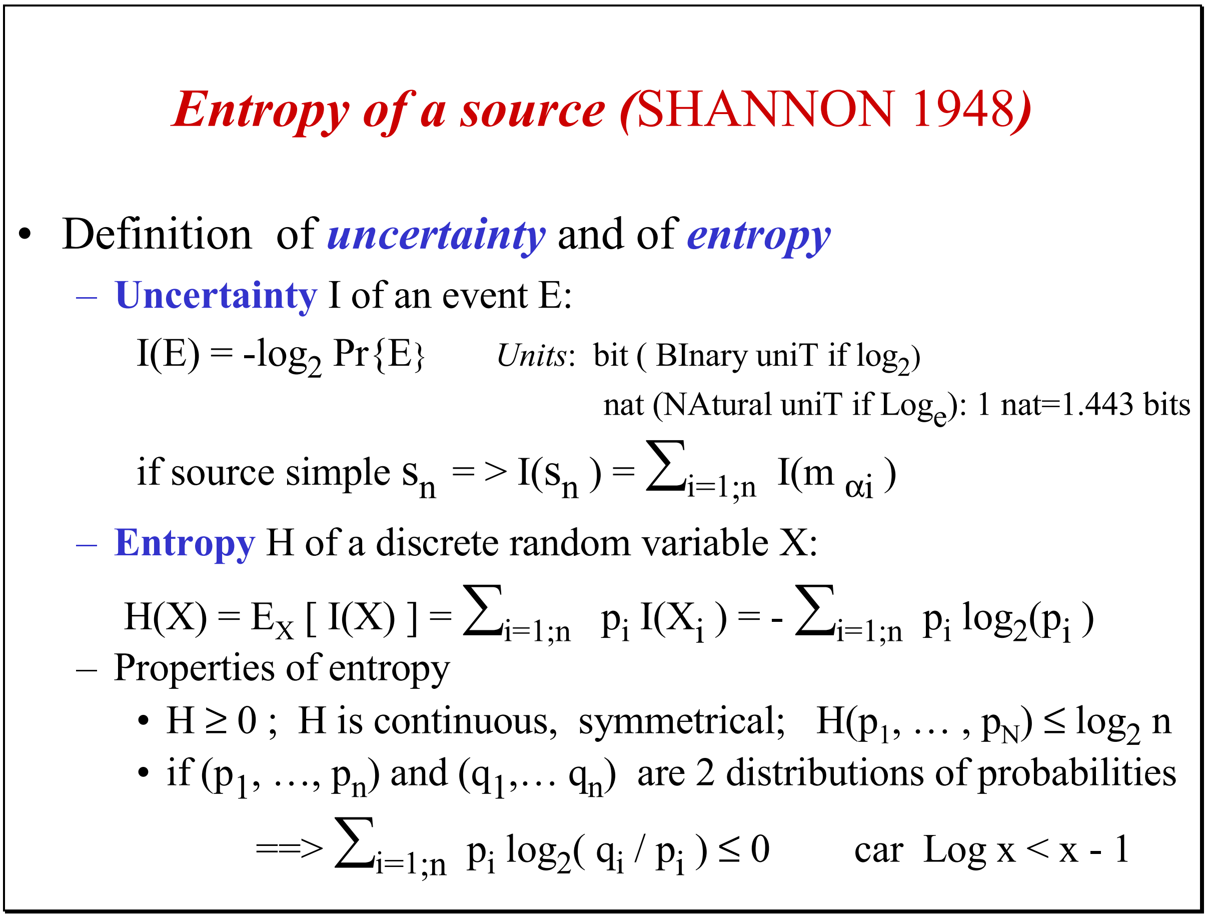
- Alphabet: finite set of symbols A = { a_1, a_2, ……, a_k }

♦ *Definitions*:

A **message** is any finite set of characters taken from an **alphabet** A: a finite set of **symbols** (for example: letters, digits etc.).

A **message source** S is the system that produces a temporal series of messages m_i, each of them taken from a set of possible messages M. M is called a message (or word) codebook. The transmitted message is in fact a text formed by the syntactic rules of elementary messages called words: M = {m_1, m_2, … }, each word is written by a fixed, finite set of symbols taken from the alphabet A.

Depending on the applications, the message sources S can use dictionaries of very different types: from messages written using characters of the alphabet, numbers, punctuation and tab marks, to visual messages where the messages are digital images where for example, each word is a pixel represented as a sequence of 8 binary symbols taken from the alphabet {0, 1} (bit).



*Entropy of a source (*SHANNON 1948*)*

- Definition of ***uncertainty*** and of ***entropy***
  - **Uncertainty** I of an event E:

    $I(E) = -\log_2 \Pr\{E\}$ *Units*: bit (BInary uniT if $\log_2$)

    nat (NAtural uniT if Log_e): 1 nat=1.443 bits

    if source simple s_n $=>$ $I(s_n) = \sum_{i=1;n} I(m_{\alpha i})$
  - **Entropy** H of a discrete random variable X:

    $H(X) = E_X[\, I(X)\,] = \sum_{i=1;n} p_i\, I(X_i) = -\sum_{i=1;n} p_i \log_2(p_i)$
  - Properties of entropy
    - $H \geq 0$; H is continuous, symmetrical; $H(p_1, \dots, p_N) \leq \log_2 n$
    - if $(p_1, \dots, p_n)$ and $(q_1, \dots q_n)$ are 2 distributions of probabilities

      $==>$ $\sum_{i=1;n} p_i \log_2(q_i / p_i) \leq 0$ car $\mathrm{Log}\, x < x - 1$

The ***uncertainty*** I of an event E of probability Pr(E) is defined by:

$$\boxed{I(E) = \log_2 \frac{1}{\Pr(E)} = -\log_2 \Pr(E)}$$

- Notes:

- if Pr(E) = 1/2 then I (E) = 1 (unitary uncertainty)
- if Pr(E) = 1 then I (E) = 0: uncertainty is null for a certain event.
- The uncertainty unit is ***bit*** (*Binary Unit*). It is not the same as the bit: Binary digit.
- We can use the natural logarithm instead of the base 2 logarithm, therefore the unit is the ***nat*** (Natural Unit = 1.443 bit).

We now consider that the events E are in fact realizations of a random discrete variable X. We define the ***entropy*** H as being the average uncertainty of the random variable X. If we consider in fact each event x_i, i ∈ [1, n], as a realization of a random variable X (i.e. X is a random variable with values in $\{ x_1, x_2, \dots, x_n \}$) :

$$H(X) = E_X \{ I(X) \} = \sum_{i=1..n} \Pr\{X = x_i\} \,.\, I(x_i) = \sum_{i=1..n} p_i.I(x_i), \text{ with } p_i = \Pr\{X = x_i\}$$

The entropy depends on the probability law of X but it is not a function of the values taken by X. It is expressed in bits (or nats) and represents the average number of bits necessary to binary encode the different realizations of X.

Now let's consider an information source S defined by a set of possible m_i (codebook): $S\{m_1, m_2, \dots, m_N\}$, and by a mechanism such as for emitting messages:

$s_n = \{m_{\alpha 1}, m_{\alpha 2}, \dots, m_{\alpha n}\}$ with $m_{\alpha 1}$: 1st emitted message, …, $m_{\alpha n}$: nth emitted message.

Warning: the index « i » in α_i defines the temporal index in the sequence of messages emitted by the source. α_i defines the index of the ith message emitted in the codebook M of possible messages, generally: $N \neq n$.

The choice of $m_{\alpha i}$ occurs according to a given probability law. The emission of a discrete source of information thus corresponds to a sequence of random variables X_i, i∈ [1, n]:

The probability of s_n can be expressed as a product of conditional probabilities:

$$\Pr(s_n) = \Pr\{X_1 = m_{\alpha 1}\} \Pr\{X_2 = m_{\alpha 2} / X_1 = m_{\alpha 1}\} \dots \Pr\{ X_n = m_{\alpha n} / X_1 = m_{\alpha 1}, \dots, X_{n-1} = m_{\alpha n-1}\}$$

In the case of simple sources, the n random variables X_i are independent and of the same law, which gives:

$$\forall (i, j) \in [1, n] \times [1, N], \Pr\{ X_i = m_j \} = p_j, \text{ et } \Pr\{s_n\} = p_{\alpha 1}.p_{\alpha 2}\dots p_{\alpha n}$$

$$\Rightarrow I (s_n) = - \log_2 \Pr\{s_n\} = - \log_2 (p_{\alpha 1}.p_{\alpha 2}\dots p_{\alpha n}) = \sum_{i=1..n} - \log_2 p_{\alpha i} = \sum_{i=1..n} I (m_{\alpha 1})$$

$$\boxed{I (s_n) = \sum_{i=1..n} I (m_{\alpha 1})}$$

In the case of a discrete source of « n » messages m_i, where each message m_i is associated with a probability p_i, the entropy H of the source S is given by:

$$\boxed{H(S) = -\sum_{i=1}^{n} p_i \,.\, \log_2 p_i}$$

- *Properties of entropy:*

- As $0 \leq p_i \leq 1$ and $\sum_{i=1}^{n} p_i = 1$, then $H(X) > 0$: the entropy is positive.
- Given $(p_1, p_2, \ldots, p_n)$ and $(q_1, q_2, \ldots, q_n)$ two probability laws, then $\sum_{i=1}^{n} p_i \log_2 \frac{q_i}{p_i} \leq 0$.

  $\forall x > 0$, we have $\mathrm{Ln}\, x \leq x - 1$ so $\ln \frac{q_i}{p_i} \leq \frac{q_i}{p_i} - 1$, being $\log_2 \frac{q_i}{p_i} \leq \frac{1}{\ln 2}(\frac{q_i}{p_i} - 1)$

  thus $\sum_{i=1}^{n} p_i \log_2 \frac{q_i}{p_i} \leq \frac{1}{\ln 2} \sum_{i=1}^{n} p_i \left(\frac{q_i}{p_i} - 1\right) = \frac{1}{\ln 2}\left(\sum_{i=1}^{n} q_i - \sum_{i=1}^{n} p_i\right) = \frac{1}{\ln 2}(1-1) = 0$
- The entropy of a random variable X with n possible values is maximal and is worth $\log_2 n$ when X follows a uniform probability law. By taking $q_1 = q_2 = \ldots = q_n = \frac{1}{n}$ (uniform law), in the previous property:

  $$\sum_{i=1}^{n} p_i \log_2 \frac{q_i}{p_i} \leq 0 \Leftrightarrow -\sum_{i=1}^{n} p_i \log_2 p_i \leq -\sum_{i=1}^{n} p_i \log_2 q_i$$

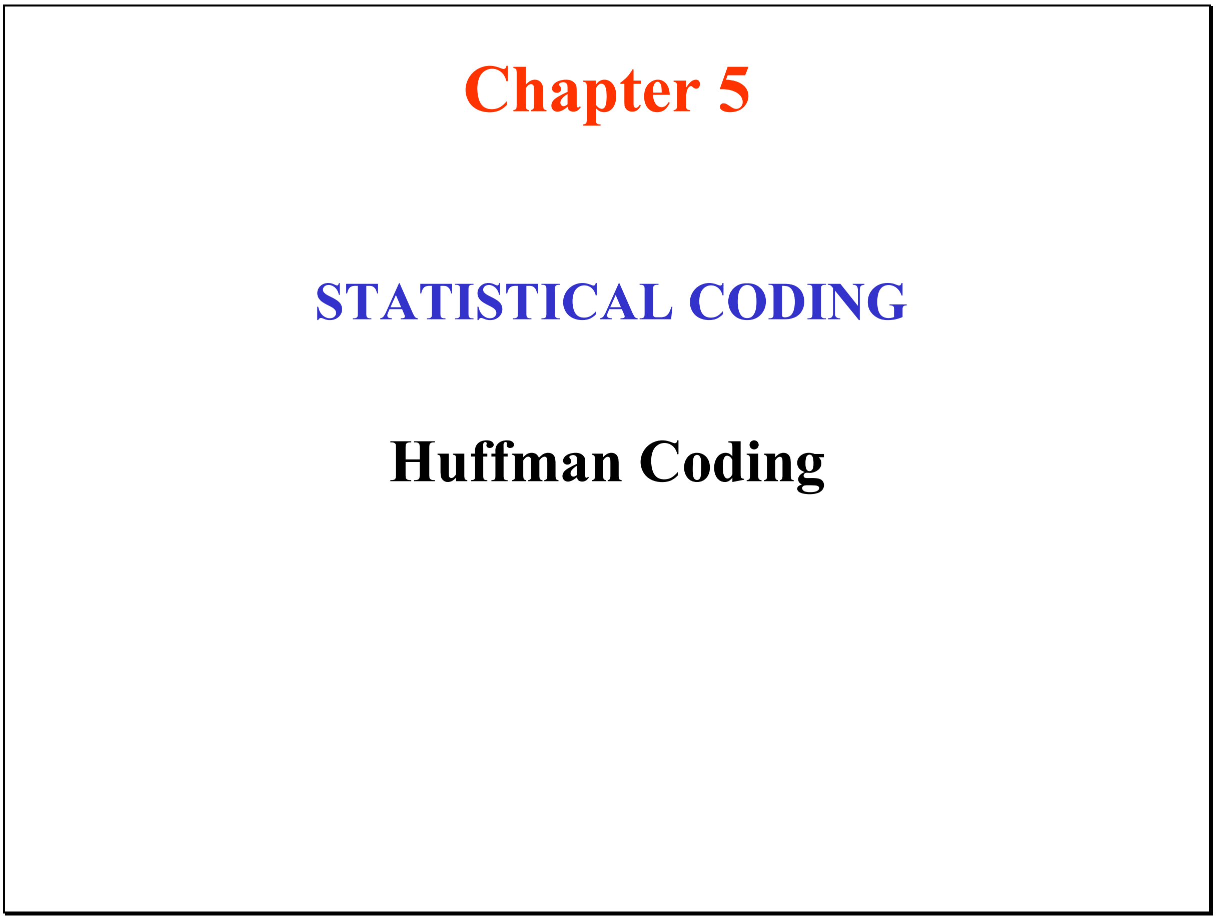
  $$\Leftrightarrow H(X) \leq -\sum_{i=1}^{n} p_i \log_2 \frac{1}{n}$$

  $$\Leftrightarrow H(X) \leq -\log_2 \frac{1}{n} \sum_{i=1}^{n} p_i = \log_2 n$$
- The entropy is continuous and symmetrical.

For the rest of this course, we will systematically use the logarithm base 2.

Simple example:

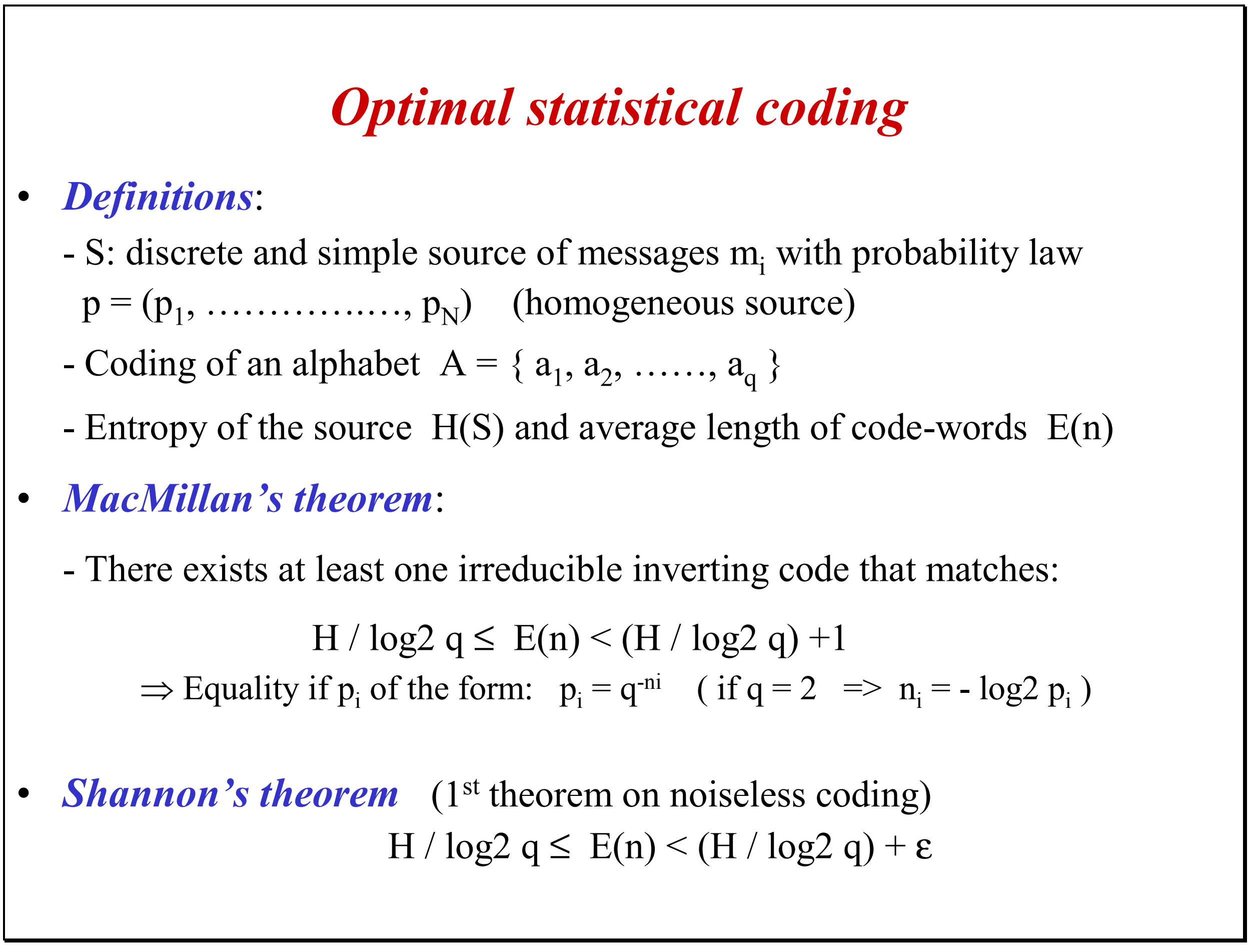
Let's consider a source S, of uniform law, that sends messages from the 26-character French (a,b,c, …, z). To this alphabet we add the "space" character as a word separator.
The alphabet is made up of 27 characters: $H(S) = -\sum_{i=1}^{27} \frac{1}{27} \log_2 \frac{1}{27} = \log_2(27) = 4.75$ bits of information per character. Actually, the entropy is close to 4 bits of information per character on a very large amount of French text.



Chapter 5

STATISTICAL CODING

Huffman Coding



Optimal statistical coding

- ***Definitions***:
  - S: discrete and simple source of messages m_i with probability law $p = (p_1, \dots\dots\dots\dots\dots, p_N)$ (homogeneous source)
  - Coding of an alphabet $A = \{ a_1, a_2, \dots\dots, a_q \}$
  - Entropy of the source H(S) and average length of code-words E(n)
- ***MacMillan's theorem***:
  - There exists at least one irreducible inverting code that matches:

$$H / \log2\, q \leq E(n) < (H / \log2\, q) + 1$$

$\Rightarrow$ Equality if p_i of the form: $p_i = q^{-ni}$ (if $q = 2$ => $n_i = -\log2\, p_i$)

- ***Shannon's theorem*** (1st theorem on noiseless coding)

$$H / \log2\, q \leq E(n) < (H / \log2\, q) + \varepsilon$$

0

In the previous resource, "Defining coding and properties" we saw that the objectives of coding are mainly to transcribe information and to reduce the quantity of symbols necessary to represent information. By optimizing the coding, we attempt to reduce the quantity of symbols as much as possible.

Let's consider a simple source of homogeneous information $S = \{m_1, m_2, \dots, m_N\}$ armed with a probability law $p = \{ p_1, p_2, \dots, p_N \}$ où $p_i = \Pr\{m = m_i\}$.
If M_i is the code-word that corresponds to the message m_i, we call $n_i = n(M_i)$ the number of characters that belong to the alphabet A (Card(A) = q) needed for the coding of m_i, n_i is thus the length of the code-word M_i.

The average length of the code-words is then: $E(n) = \sum_{i=1}^{N} p_i\, n_i$.

The average uncertainty of a source is the entropy H. The average uncertainty per character of the alphabet A is equal to $\frac{H}{E(n)}$, so we get: $\frac{H}{E(n)} \leq \log_2 q$ because alphabet A contains « q » characters. From this inequality, we deduce that $E(n) \geq \frac{H}{\log_2 q}$.

To optimize coding, we want to reduce E(n), the average length of the code-words. This average length cannot be lower than $\frac{H}{\log_2 q}$. Nevertheless, two theories show that it is possible to obtain equality $E(n) = \frac{H}{\log_2 q}$ and an optimal code:

- ***Mac Millan's theorem:***

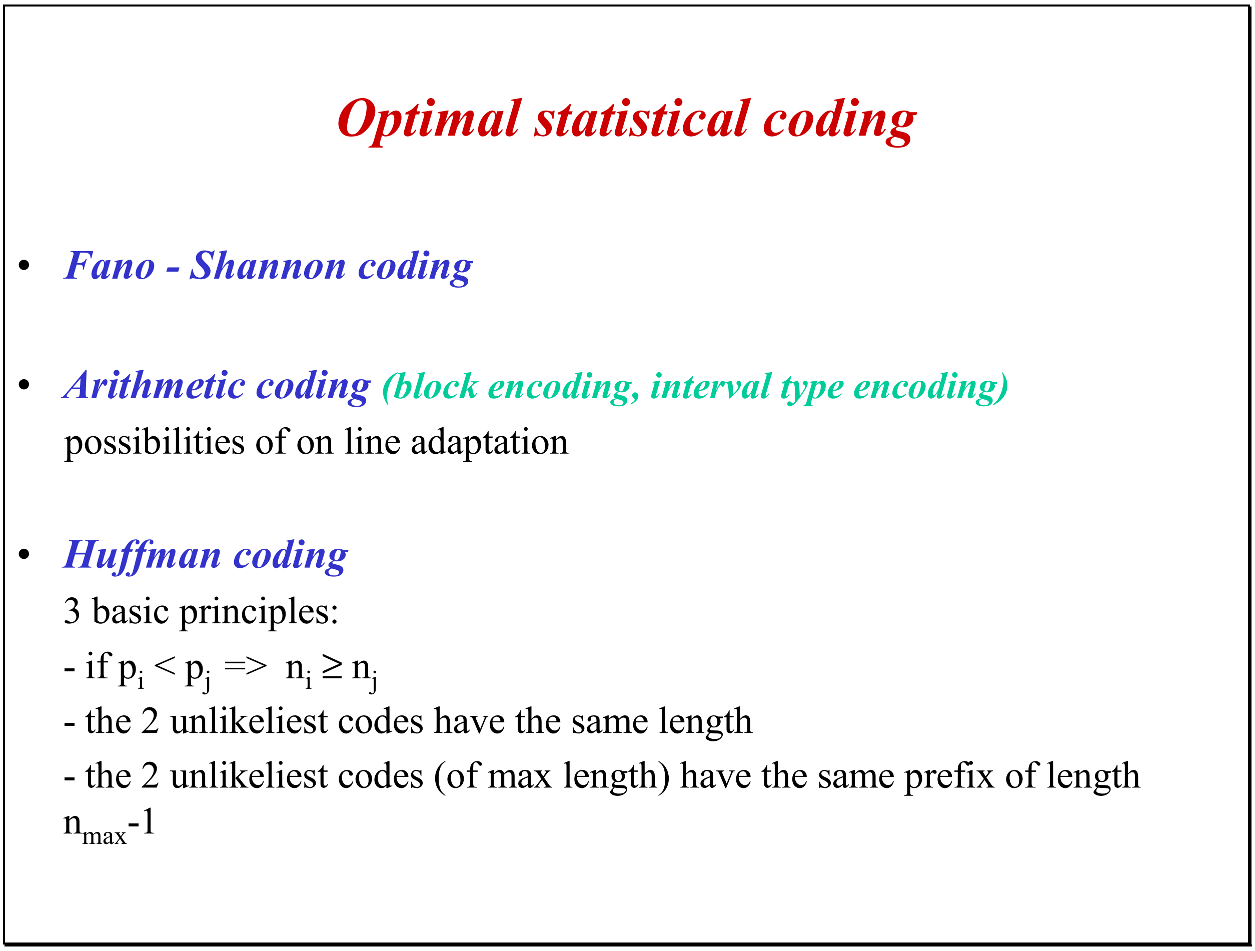
An information source S with entropy H coded by an inverting way with an alphabet counting q characters is such that: $E(n) \geq \frac{H}{\log_2 q}$ and there exists at least one irreducible code of a given law such as $\leq \frac{H}{\log_2 q} + 1$. Equality is reached if $p_i = q^{-ni}$ (i.e. $n_i = - \log_q p_i$) and we then have an optimal coding.

Note:

In the particular case of a binary alphabet {0, 1}, we have q = 2. If the relationship $n_i = - \log_2 p_i$ is true, we then have E(n) = H: the entropy is the bottom limit of the set of code-word average lengths and this lower limit is reached.

- ***Shannon's theorem of noiseless coding:***

Any homogeneous source of information is such as there exists an irreducible coding for which the average length of code-words is as close as we want to the lower limit $\frac{H}{\log_2 q}$. The demonstration of this theorem uses irreducible coding in blocks (block coding assigns a code-word to each block of « k » messages of S, consecutive or not).



Optimal statistical coding

- ***Fano - Shannon coding***
- ***Arithmetic coding*** ***(block encoding, interval type encoding)***
  possibilities of on line adaptation
- ***Huffman coding***
  3 basic principles:
  - if $p_i < p_j \Rightarrow n_i \geq n_j$
  - the 2 unlikeliest codes have the same length
  - the 2 unlikeliest codes (of max length) have the same prefix of length n_{max}-1

The presentation above shows three types of codes that are close to optimal code:

- ***Shannon-Fano coding:***

It tries to approach as much as possible the most compact irreducible code, but the probability p_i is not usually equal to 2^{-ni}, so the coding can only be close to optimal coding.
The probabilities p_i associated with the messages m_i are arranged by decreasing order then we fix n_i so that: $n_i \geq \log_2 \frac{1}{p_i} \geq n_i - 1$. Finally, we choose each code-word M_i of length n_i so that none of the previously chosen code-words forms a prefix (it avoids decoding ambiguity cf.: *"classification of the codes"*).

- ***Arithmetic coding:***

The code is associated with a sequence of messages from the source and not with each message. Unlike Huffman coding, which must have an integer length of bits per message and which does not always allow an optimal compression, arithmetic coding lets you code a message on a non-integer number of bits: this is the most effective method, but it is also the slowest. The different aspects of this coding are more fully developed in the resource: "*Statistical coding: arithmetic coding*".

♦ ***Huffman coding:***

Huffman coding is the optimal irreducible code. It is based on three principles:

- if $p_j > p_i$ then $n_i \leq n_j$,
- the two most unlikely words have equal lengths,
- the latters are written with the same n_{max}-1 first characters.

By using this procedure iteratively, we build the code-words M_i of the messages m_i.

- <u>Example</u>:

Let be a source S = {m_1, m_2, …, m_8} with a probability law: p_1 = 0,4 ; p_2 = 0,18 ; p_3 = p_4 = 0,1 ; p_5 = 0,07 ; p_6 = 0,06 ; p_7 = 0,05 ; p_8 = 0,04.

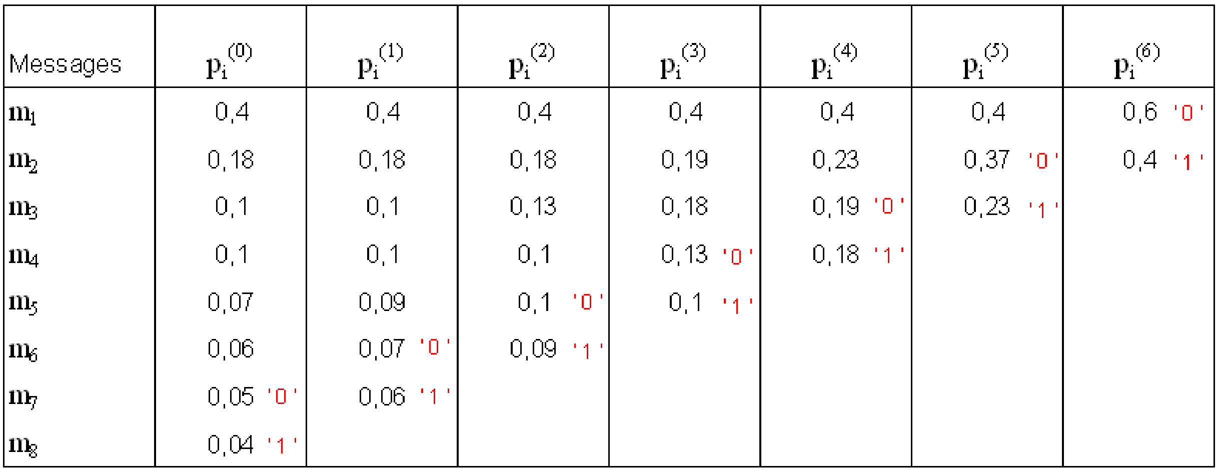
We place these probabilities in decreasing order in the column $p_i^{(0)}$ of the table below. We can see that in the column $p_i^{(0)}$ the probabilities of the messages m_7 and m_8 are the smallest, we add them and reorder the probabilities, still in decreasing order, to create the column $p_i^{(1)}$:

Messages	$p_i^{(0)}$	$p_i^{(1)}$
m_1	0,4	0,4
m_2	0,18	0,18
m_3	0,1	0,1
m_4	0,1	0,1
m_5	0,07	0,09
m_6	0,06	0,07
m_7	0,05	0,06
m_8	0,04	

Generally, we add the two smallest probabilities in the column $p_i^{(k)}$, then we reorder the probabilities in decreasing order to obtain the column $p_i^{(k+1)}$. Finally we get the following table:

Messages	$p_i^{(0)}$	$p_i^{(1)}$	$p_i^{(2)}$	$p_i^{(3)}$	$p_i^{(4)}$	$p_i^{(5)}$	$p_i^{(6)}$
m_1	0,4	0,4	0,4	0,4	0,4	0,4	0,6
m_2	0,18	0,18	0,18	0,19	0,23	0,37	0,4
m_3	0,1	0,1	0,13	0,18	0,19	0,23	
m_4	0,1	0,1	0,1	0,13	0,18		
m_5	0,07	0,09	0,1	0,1			
m_6	0,06	0,07	0,09				
m_7	0,05	0,06					
m_8	0,04						

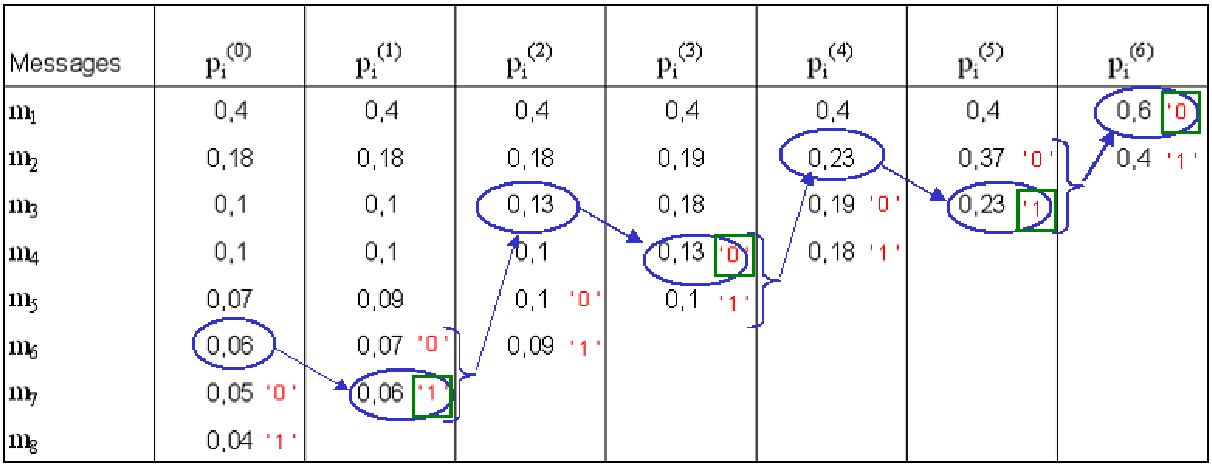
We assign the bits ‘0’ and ‘1’ to the last two elements of each column:



Messages	$p_i^{(0)}$	$p_i^{(1)}$	$p_i^{(2)}$	$p_i^{(3)}$	$p_i^{(4)}$	$p_i^{(5)}$	$p_i^{(6)}$
m_1	0,4	0,4	0,4	0,4	0,4	0,4	0,6 '0'
m_2	0,18	0,18	0,18	0,19	0,23	0,37 '0'	0,4 '1'
m_3	0,1	0,1	0,13	0,18	0,19 '0'	0,23 '1'	
m_4	0,1	0,1	0,1	0,13 '0'	0,18 '1'		
m_5	0,07	0,09	0,1 '0'	0,1 '1'			
m_6	0,06	0,07 '0'	0,09 '1'				
m_7	0,05 '0'	0,06 '1'					
m_8	0,04 '1'						

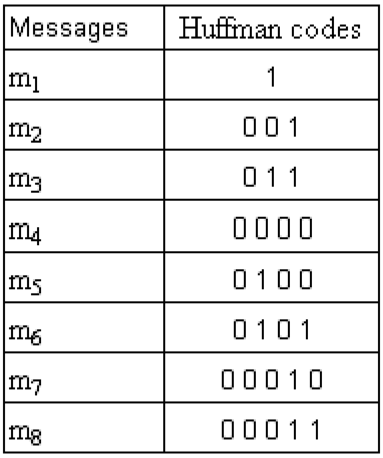
For each message m_i, we go through the table from left to right and in each column we can see the associated probability $p_i^{(k)}$ (blue path on the illustration below).
The code-word M_i is then obtained by starting from the last column on the right and moving back to the first column on the left, by selecting the bits associated with the probabilities $p_i^{(k)}$ of the message m_i (green rectangles on the illustration below).

For example, we want to determine the code-word M_6 of the message m_6. We detect all the probabilities $p_6^{(k)}$:



Messages	$p_i^{(0)}$	$p_i^{(1)}$	$p_i^{(2)}$	$p_i^{(3)}$	$p_i^{(4)}$	$p_i^{(5)}$	$p_i^{(6)}$
m_1	0,4	0,4	0,4	0,4	0,4	0,4	0,6 '0'
m_2	0,18	0,18	0,18	0,19	0,23	0,37 '0'	0,4 '1'
m_3	0,1	0,1	0,13	0,18	0,19 '0'	0,23 '1'	
m_4	0,1	0,1	0,1	0,13 '0'	0,18 '1'		
m_5	0,07	0,09	0,1 '0'	0,1 '1'			
m_6	0,06	0,07 '0'	0,09 '1'				
m_7	0,05 '0'	0,06 '1'					
m_8	0,04 '1'						

The code-word M_6 is thus obtained by simply reading from right to left the bits contained in the green rectangles: ‘0’ – ‘1’ – ‘0’ – ‘1’. By following the same procedure for each message, we obtain:



Messages	Huffman codes
m_1	1
m_2	0 0 1
m_3	0 1 1
m_4	0 0 0 0
m_5	0 1 0 0
m_6	0 1 0 1
m_7	0 0 0 1 0
m_8	0 0 0 1 1

The average length of the code-words is equal to:

$$E(n) = \sum_{i=1}^{8} p_i\, n_i = 0{,}4\times1 + 0{,}18\times3 + 0{,}1\times3 + 0{,}1\times4 + 0{,}07\times4 + 0{,}06\times4 + 0{,}05\times5 + 0{,}04\times5$$

$$E(n) = 2{,}61$$

We can compare this size with the entropy H of the source:

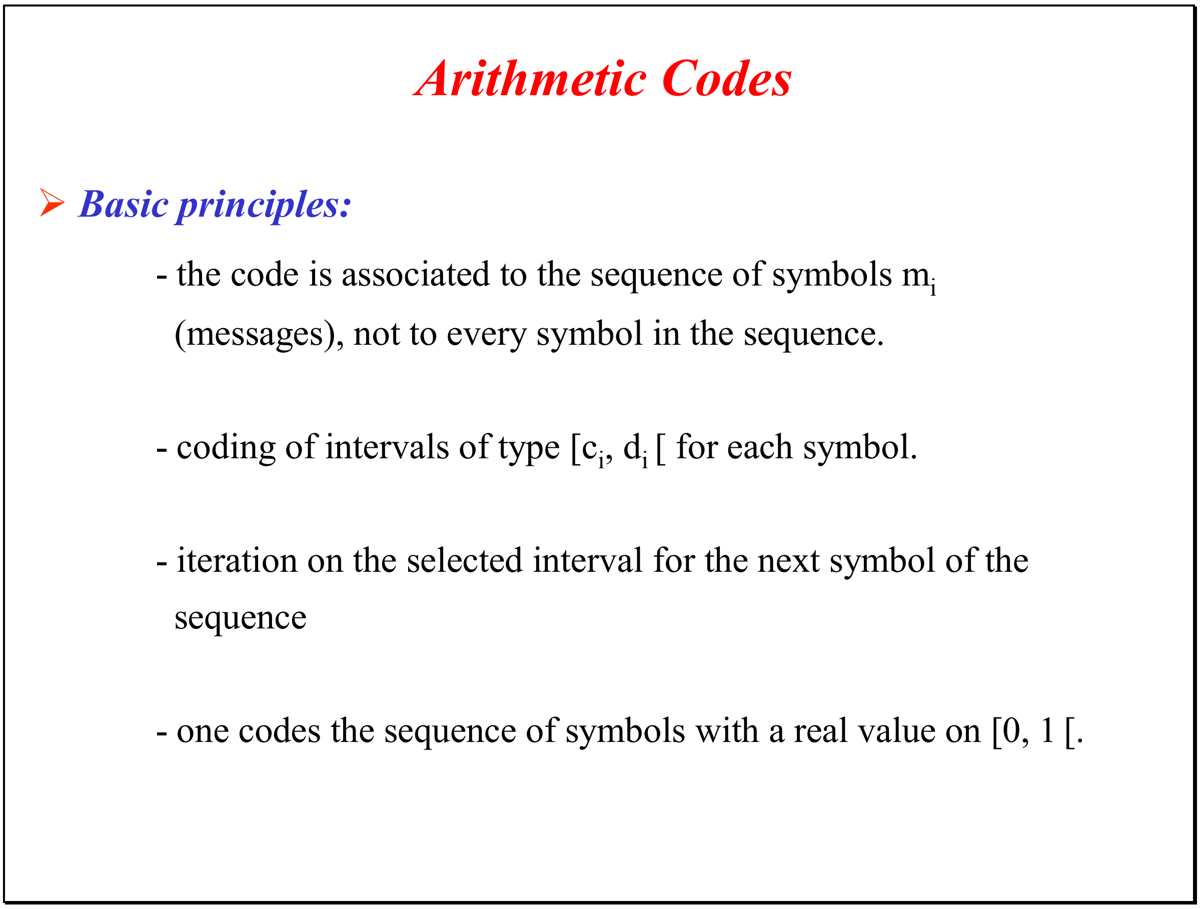
$$H = -\sum_{i=1}^{8} p_i\ .\log_2 p_i = 2{,}552$$

The efficiency η of the Huffman coding for this example is thus $\frac{2{,}552}{2{,}61}$ = 97.8 %. For comparison purposes, 3 bits are needed to code 8 different messages with a natural binary (2^3 = 8). For this example, the efficiency of the natural binary coding is only $\frac{2{,}552}{3}$ = 85 %.

Chapter 5

STATISTICAL CODING

Arithmetic Coding



Arithmetic Codes

➢ ***Basic principles:***

- the code is associated to the sequence of symbols m_i (messages), not to every symbol in the sequence.

- coding of intervals of type $[c_i, d_i[$ for each symbol.

- iteration on the selected interval for the next symbol of the sequence

- one codes the sequence of symbols with a real value on [0, 1 [.

Arithmetic codes allow you to encode a sequence of events by using estimates of the probabilities of the events. The arithmetic code assigns one codeword to each possible data set. This technique differs from Huffman codes, which assign one codeword to each possible event.

The codeword assigned to one sequence is a real number which belongs to the half-open unit interval [0, 1 [. Arithmetic codes are calculated by successive subdivisions of this original unit interval. For each new event in the sequence, the subinterval is refined using the probabilities of all the individual events.

Finally we create a half-open subinterval of [0, 1[, so that any value in this subinterval encodes the original sequence.

Arithmetic Codes

➢ *Definitions:*

- Let $S = \{s_1, s_2, \ldots, s_N\}$ be a source and $p_k = \Pr(s_k)$

- $[Ls_k, Hs_k[$ is the interval corresponding to the symbol s_k
  with: $Hs_k - Ls_k = p_k$

➢ *Encoding algorithm:*

1) Initialization: $L_c = 0$; $H_c = 1$
2) Calculate code sub-intervals
3) Get next input symbol s_k
4) Update the code sub-interval
5) Repeat from step 2 until all the sequence has been encoded

Let $S = \{s_1, \ldots, s_N\}$ be a source which can produce N source symbols. The probabilities of the source symbols are denoted by: $\forall \in [1, N], P\{s_k\} = p_k$.
Here is the basic algorithm for the arithmetic coding of a sequence $s_M = \{s_{\alpha 1}, s_{\alpha 2}, \ldots, s_{\alpha M}\}$ of M source symbols ($s_{\alpha k}$ stands for the k-th source symbol that occurs in the sequence we want to encode):

- Step1:

Let us begin with a current half-open interval $[L_c, H_c[$ initialized to $[0, 1[$ (this interval corresponds to the probability of choosing the first source symbol $s_{\alpha 1}$). The length of the current interval is thus defined by: length = $H_c - L_c$.

- Step 2:

For each source symbol in the sequence, we subdivide the current interval into half-open subintervals $[Ls_k, Hs_k[$, one for each possible source symbol s_k.
The size of a symbol's subinterval $[Ls_k, Hs_k[$ depends on the probability p_k of the symbol s_k.
The subinterval length ($Ls_k - Hs_k$) is defined so that: $H_k - L_k = p_k$, then:

$$Ls_k = L_c + \text{length} \times \sum_{i=1}^{k-1} p_i \quad \text{and} \quad Hs_k = L_c + \text{length} \times \sum_{i=1}^{k} p_i \ .$$

- Step 3:

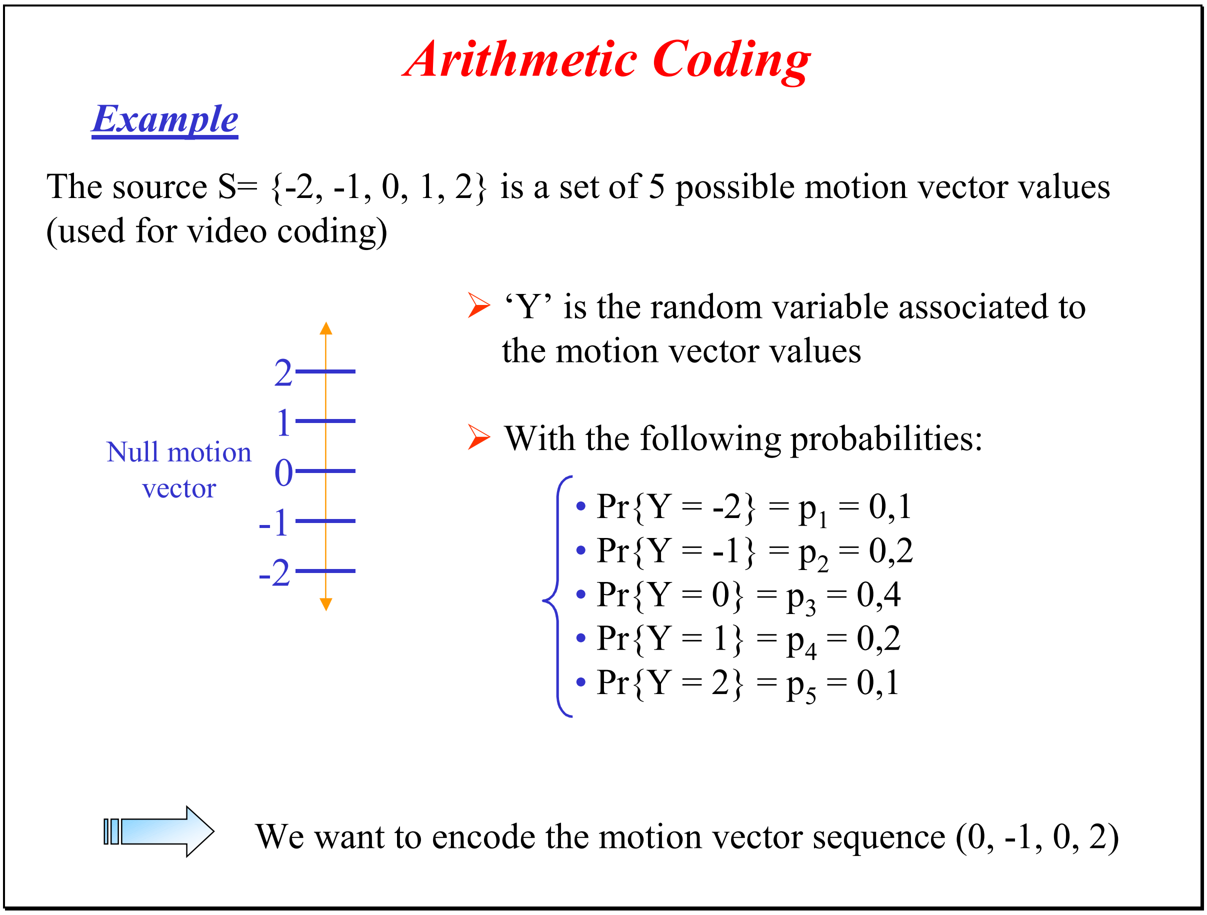
We select the subinterval corresponding to the source symbol s_k that occurs next and make it the new current interval $[L_c, H_c[$: $\begin{cases} L_c = L_c + \text{length} \times Ls_k \\ H_c = L_c + \text{length} \times Hs_k \end{cases}$

- Step 4 :

This new current interval is subdivided again as described in Step 2.

- Step 5 :

Repeat the steps 2, 3, and 4 until the whole sequence of source symbols has been encoded.



Arithmetic Coding

Example

The source S= {-2, -1, 0, 1, 2} is a set of 5 possible motion vector values (used for video coding)

2
1
Null motion vector 0
-1
-2

- 'Y' is the random variable associated to the motion vector values
- With the following probabilities:

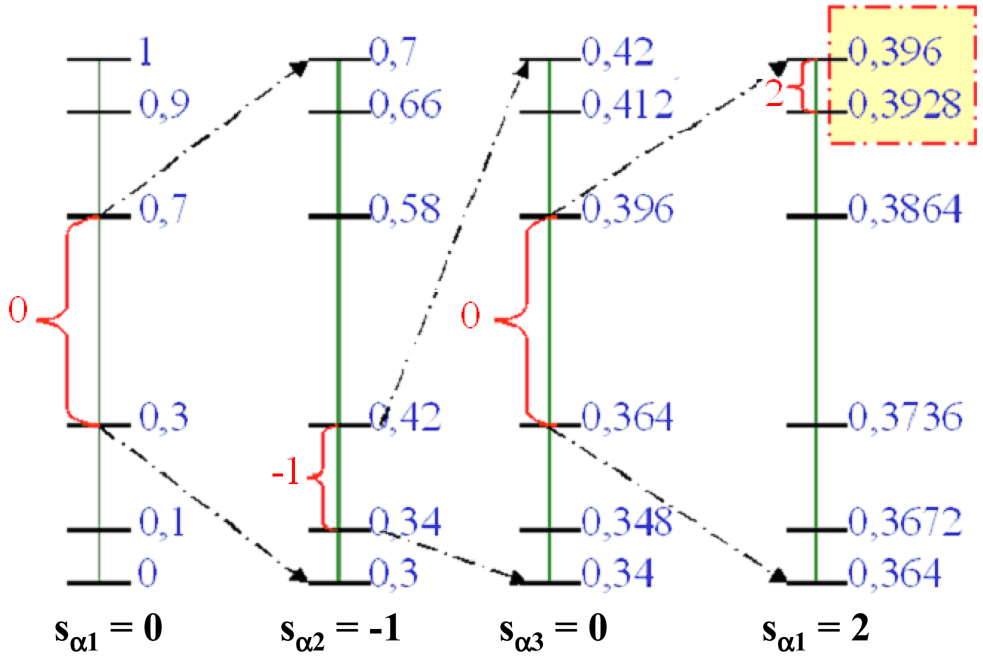
$$\begin{cases} \Pr\{Y = -2\} = p_1 = 0{,}1 \\ \Pr\{Y = -1\} = p_2 = 0{,}2 \\ \Pr\{Y = 0\} = p_3 = 0{,}4 \\ \Pr\{Y = 1\} = p_4 = 0{,}2 \\ \Pr\{Y = 2\} = p_5 = 0{,}1 \end{cases}$$

We want to encode the motion vector sequence (0, -1, 0, 2)

Here is an example for encoding motion vectors of a video signal with an arithmetic code.

Arithmetic Coding

Subdivisions of the current sub-intervals

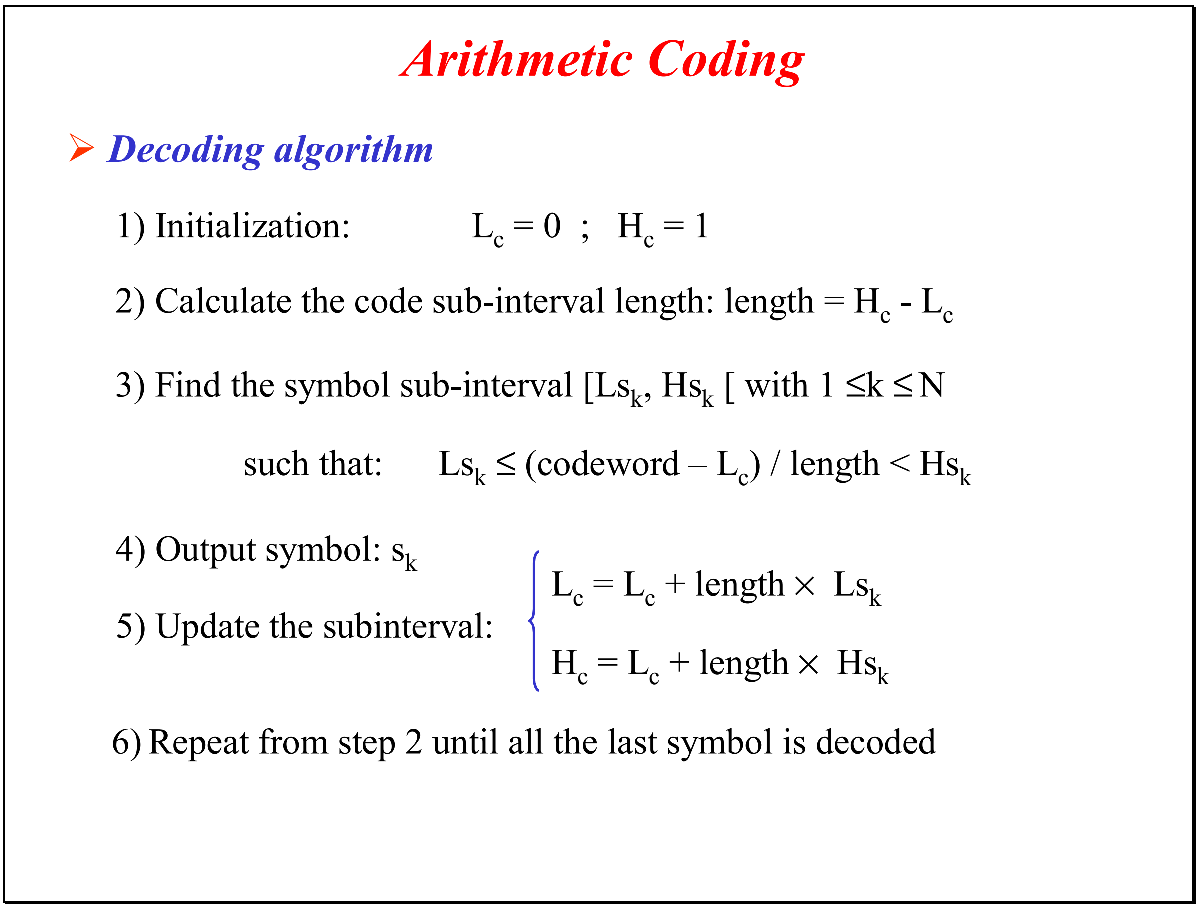


To encode the sequence $\{s_{\alpha1}, s_{\alpha2}, s_{\alpha3}, s_{\alpha4}\} = \{s_3, s_2, s_3, s_5\} = \{0, -1, 0, 2\}$, we subdivide the unit current interval [0, 1[into 5 half-open intervals, then we select the subinterval corresponding to the first motion vector that occurs in the sequence (value 0). This subinterval is the new current interval. We subdivide it and we select the subinterval corresponding to the next event (the motion vector -1).

We repeat these steps for each source symbol of the sequence (here these source symbols are the motion vectors). Consequently, we can encode the sequence (0, -1, 0, 2) of vertical motion vectors by any value in the half-open range [0.3928, 0.396[. The value 0.3945 encodes this sequence; therefore we need 8 bits to encode this sequence:

$$0.3945 = 0\times2^{-1} + 2^{-2} + 2^{-3} + 0\times2^{-4} + 0\times2^{-5} + 2^{-6} + 0\times2^{-7} + 2^{-8}$$

So we need 8/5 = 1.6 bits/symbol.



Arithmetic Coding

➤ Decoding algorithm

1) Initialization: $L_c = 0$; $H_c = 1$

2) Calculate the code sub-interval length: length = $H_c - L_c$

3) Find the symbol sub-interval $[Ls_k, Hs_k[$ with $1 \leq k \leq N$

such that: $Ls_k \leq (\text{codeword} - L_c) / \text{length} < Hs_k$

4) Output symbol: s_k

5) Update the subinterval: $$\begin{cases} L_c = L_c + \text{length} \times Ls_k \\ H_c = L_c + \text{length} \times Hs_k \end{cases}$$

6) Repeat from step 2 until all the last symbol is decoded

The figure above describes the decoding algorithm of a codeword obtained after having encoded a sequence of source symbols with an arithmetic code. This decoding algorithm is performed for the previous example. Let us consider the codeword $M_c = 0.3945$:

[Algorithm beginning]

Step 1:

We initialize the current interval $[L_c, H_c[$: $L_c = 0$ et $H_c = 1$.

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 1$.

Step 3:

We calculate the value $(M_c - L_c) / L = 0.3945$, and we select the subinterval $[Ls_k, Hs_k[$ so that $M_c \in [Ls_k, Hs_k[$.
Here, the selected subinterval is [0.3, 0.7[. This subinterval corresponds to the half-open interval $[Ls_3, Hs_3[$.

Step 4:

The ***first symbol*** $\boldsymbol{s_{\alpha 1}}$ of the sequence is thus s_3 (***motion vector 0***).

Step 5:

We create the new current interval $[L_c, H_c[$ for encoding the next source symbol:

$$L_c = L_c + L \times Ls_3 = 0 + 1 \times 0.3 = 0.3$$
$$H_c = L_c + L \times Hs_3 = 0 + 1 \times 0.7 = 0.7$$

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.7 – 0.3 = 0.4$.

Step 3:

$(M_c - L_c) / L = (0.3945 – 0.3) / 0.4 = 0,2363$.
This value belongs to the subinterval $[Ls_2, Hs_2[= [0.1, 0.3[$.

Step 4:

The ***second symbol*** $\boldsymbol{s_{\alpha 2}}$ of the sequence is thus $\boldsymbol{s_2}$ (***motion vector -1***).

Step 5:

$L_c = L_c + L \times Ls_2 = 0.3 + 0.4 \times 0.1 = 0.34$
$H_c = L_c + L \times Hs_2 = 0.3 + 0.4 \times 0.3 = 0.42$
Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.42 – 0.34 = 0.08$.

Step 3:

$(M_c - L_c) / L = (0.3945 – 0.34) / 0.08 = 0.6812$.
This value belongs to the subinterval $[Ls_3, Hs_3[= [0.3, 0.7[$.

Step 4:

The ***third symbol*** $\boldsymbol{s_{\alpha 3}}$ of the sequence is thus $\boldsymbol{s_3}$ (***motion vector 0***).

Step 5:

$L_c = L_c + L \times Ls_3 = 0.34 + 0.08 \times 0.3 = 0.364$
$H_c = L_c + L \times Hs_3 = 0.34 + 0.08 \times 0.7 = 0.396$

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.396 – 0.364 = 0.032$.

Step 3:

$(M_c - L_c) / L = (0.3945 – 0.364) / 0.032 = 0.9531$.
This value belongs to the subinterval $[Ls_5, Hs_5[= [0.9, 1[$.

Step 4:

The ***fourth symbol*** $\boldsymbol{s_{\alpha 4}}$ of the sequence is thus $\boldsymbol{s_5}$ (***motion vector 2***).

[Algorithm end]

The decoding of the value 0.3945 allows us to rebuild the original sequence $\{s_{\alpha 1}, s_{\alpha 2}, s_{\alpha 3}, s_{\alpha 4}\} = \{s_3, s_2, s_3, s_5 \} = \{ 0, -1, 0, 2 \}$.

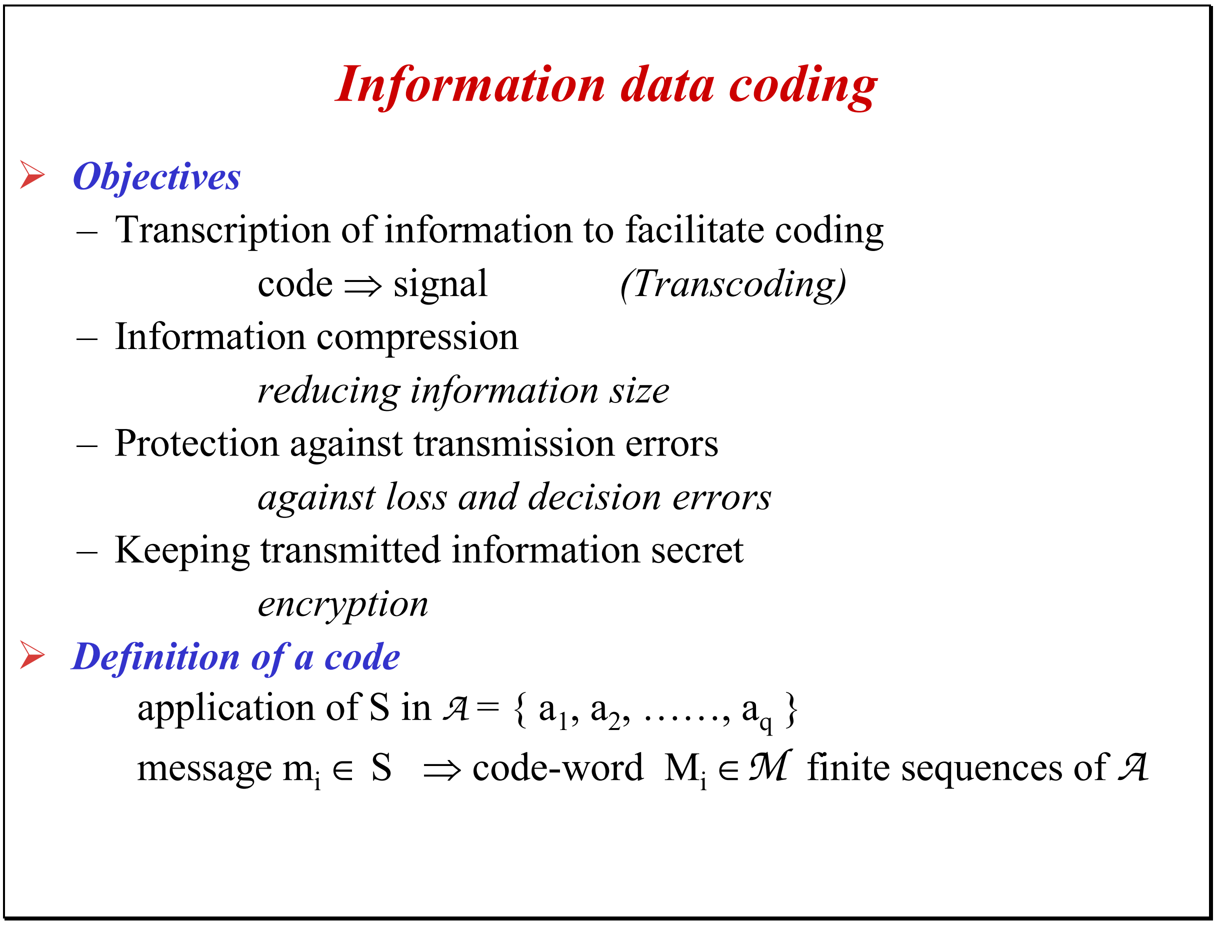
Contrary to Huffman codes, arithmetic codes allow you to allocate fractional bits to symbols. The data compression with arithmetic codes is thus more efficient. However arithmetic coding is slower than Huffman. It is not possible to start decoding without the entire sequence of symbols, which is possible in Huffman coding.
The compression rate can also be increased by using probability models which are not static. The probabilities are adapted according to the current and the previous sequences: arithmetic coding can thus handle adaptive coding.

Chapter 5

STATISTICAL CODING

Presentation

Classification of the statistical codes



Information data coding

- ***Objectives***
  - Transcription of information to facilitate coding

    code ⇒ signal *(Transcoding)*
  - Information compression

    reducing information size
  - Protection against transmission errors

    against loss and decision errors
  - Keeping transmitted information secret

    encryption
- ***Definition of a code***

  application of S in $\mathcal{A} = \{ a_1, a_2, \ldots\ldots, a_q \}$

  message $m_i \in S \Rightarrow$ code-word $M_i \in \mathcal{M}$ finite sequences of $\mathcal{A}$

Information coding consists of transcribing messages from an information source in the form of a sequence of characters taken from a predefined alphabet. The objectives of coding fall into four main categories:

- ***transcribing*** information in a form that makes it easy to create a signal that can handle the information, or easy to handle the information automatically. To do this, different codes for representing the information are used depending on the envisaged application, with transcoding operations frequently being used;

- ***reducing*** the number of information symbols needed to represent the information (in terms of the total number of symbols used): this is a space-saving role;

- ***preventing*** quality loss (distortion, noise) caused by the transmission channel and which lead to errors when you reconstruct the information when it leaves the transmission channel (upon reception);

- ***protecting*** confidential information by making it unintelligible except for its intended recipient.

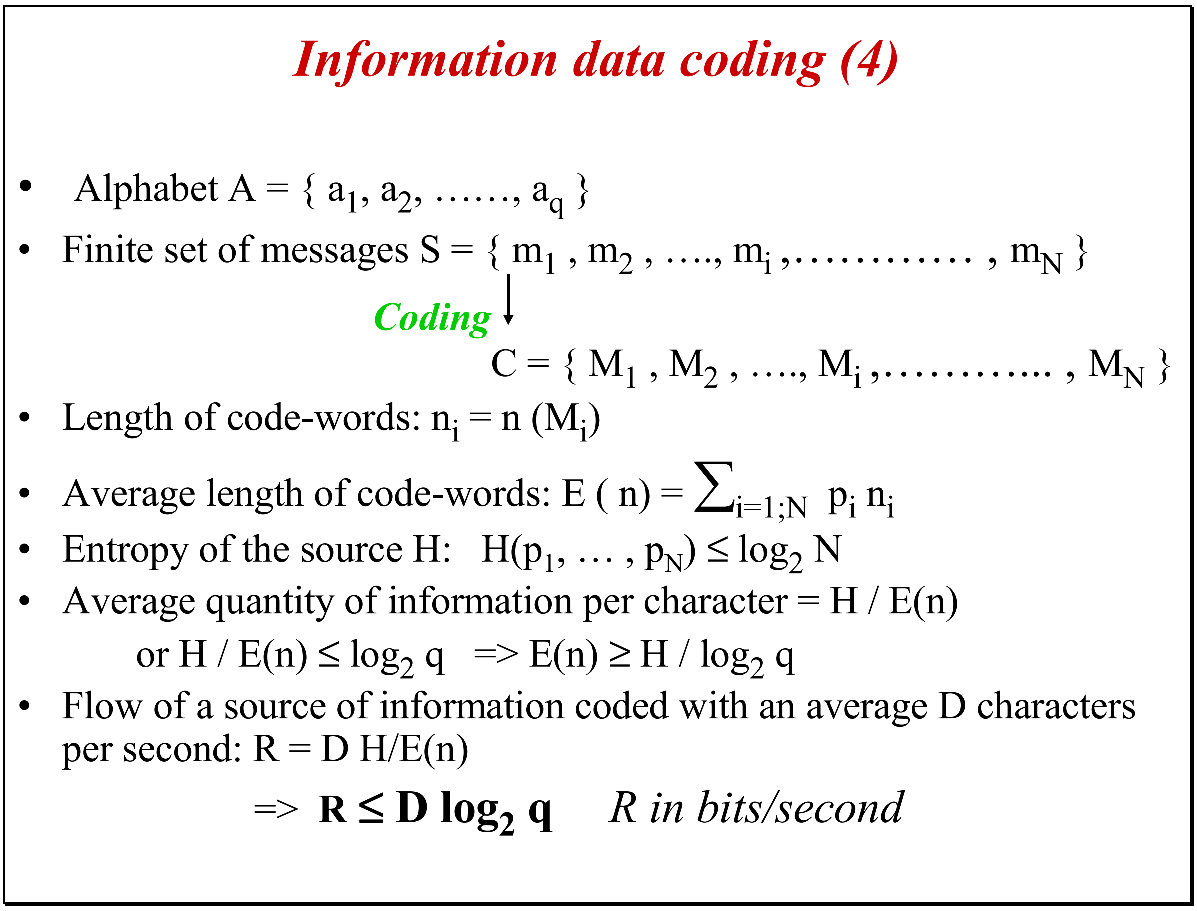
- *Definition of a code:*

Given a set, called alphabet A made up of q characters a_i : $\mathcal{A} = \{ a_1, a_2, \ldots, a_q \}$and $\mathcal{M}$ the finite set of finite sequences M_i of characters (for example: $M_i = a_{10}\ a_4\ a_7$).
Given a finite set of messages emitted by a message source S: $S=\{ m_1, \ldots, m_N \}$.

A code refers to any application of S in $\mathcal{A}$: coding of S through the use of the alphabet A.

The element M_i of $\mathcal{M}$ which corresponds to the message m_i of S is called the ***codeword*** of m_i.
Its ***length***, noted as n_i, is the number of characters belonging to $\mathcal{A}$ which compose M_i.

The ***decoding*** of a sequence of sent messages m_i involves being able to separate the codewords in a received sequence of codewords M_i. This is why we sometimes use a special spacing character in an alphabet.



Information data coding (4)

- Alphabet A = { $a_1, a_2, \ldots\ldots, a_q$ }
- Finite set of messages S = { $m_1 , m_2 , \ldots, m_i , \ldots\ldots\ldots\ldots , m_N$ }

$$S \xrightarrow{\textit{Coding}} C = \{ M_1 , M_2 , \ldots, M_i , \ldots\ldots\ldots\ldots , M_N \}$$

- Length of code-words: $n_i = n\,(M_i)$
- Average length of code-words: $E\,(n) = \sum_{i=1;N} p_i\, n_i$
- Entropy of the source H: $H(p_1, \ldots , p_N) \leq \log_2 N$
- Average quantity of information per character = H / E(n)

  or $H / E(n) \leq \log_2 q$ => $E(n) \geq H / \log_2 q$
- Flow of a source of information coded with an average D characters per second: R = D H/E(n)

  => $\mathbf{R \leq D \log_2 q}$ *R in bits/second*

From here on, we shall call the **messages** produced by the information source $\mathbf{m_i}$ and $\mathbf{M_i}$ the **codewords** associated with them.

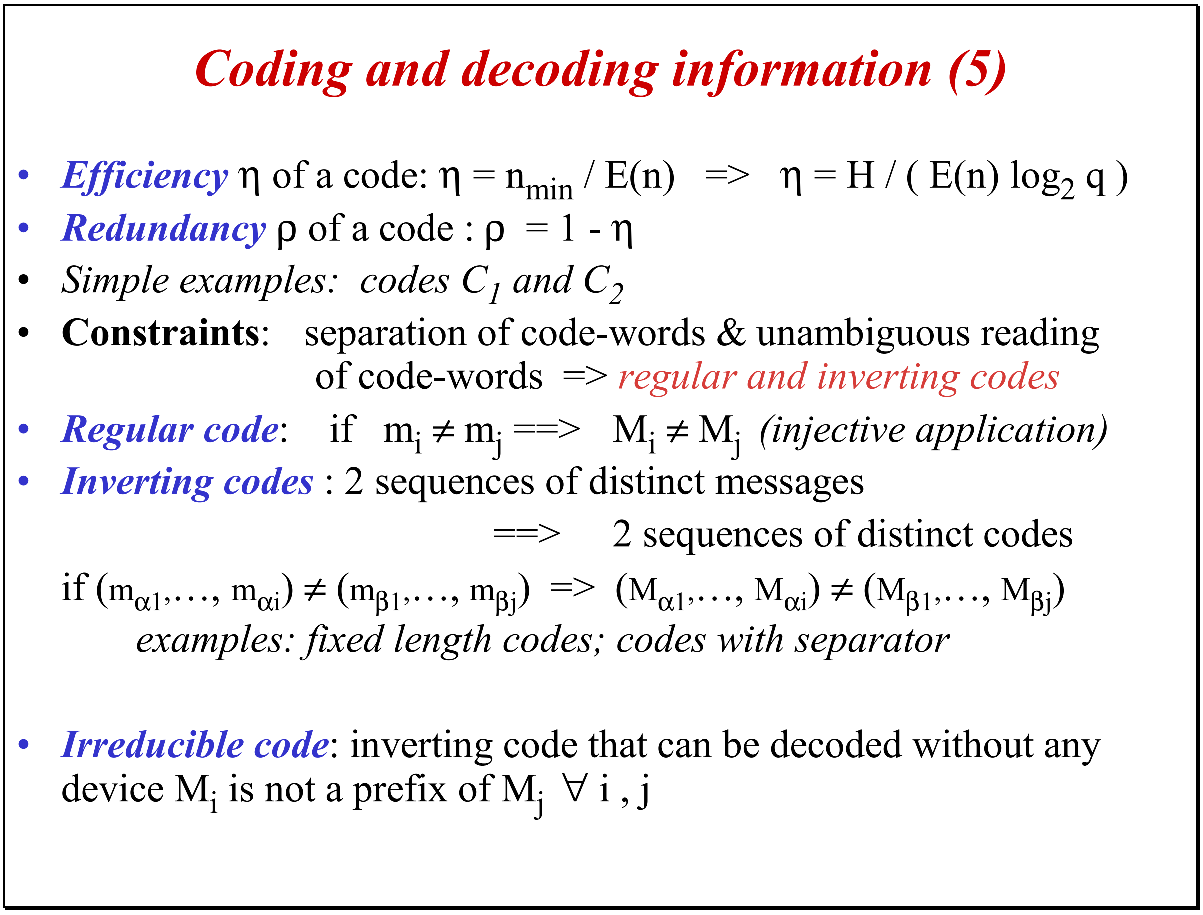
We will call $\mathbf{n_i}$ = $n(M_i)$ the number of characters belonging to an **alphabet** $\mathcal{A}$ (Card($\mathcal{A}$) = q) necessary for coding m_i, n_i being the **length** of the codeword M_i. If the source uses N possible different messages, the average length of the codewords is given by:

$$E(n) = \sum_{i=1}^{N} p_i\, n_i \text{ , where } p_i = \Pr\{ m_i \}.$$

H is the average uncertainty (i.e. the entropy) of the source S **per message** sent, so the average uncertainty (i.e. the entropy) **per character** (of the alphabet $\mathcal{A}$) equals $\frac{H}{E(n)}$ and we have: $\frac{H}{E(n)} \leq \log_2 q$ (because we have q characters in the alphabet $\mathcal{A}$), so: $E(n) \geq \frac{H}{\log_2 q}$.

Finally, if the coded information source produces D characters per second taken from the alphabet $\mathcal{A}$, $\frac{H}{E(n)}$ being the average information transported per character in bit/character, the character rate R of information is: $R = D.\frac{H}{E(n)}$.

This character rate is then limited by: $R \leq D.\log_2 q$.



Coding and decoding information (5)

- ***Efficiency*** η of a code: $\eta = n_{min} / E(n)$ => $\eta = H / (E(n) \log_2 q)$
- ***Redundancy*** ρ of a code : $\rho = 1 - \eta$
- *Simple examples: codes C_1 and C_2*
- **Constraints**: separation of code-words & unambiguous reading of code-words => *regular and inverting codes*
- ***Regular code***: if $m_i \neq m_j$ ==> $M_i \neq M_j$ *(injective application)*
- ***Inverting codes*** : 2 sequences of distinct messages ==> 2 sequences of distinct codes

  if $(m_{\alpha 1},\ldots, m_{\alpha i}) \neq (m_{\beta 1},\ldots, m_{\beta j})$ => $(M_{\alpha 1},\ldots, M_{\alpha i}) \neq (M_{\beta 1},\ldots, M_{\beta j})$

  examples: fixed length codes; codes with separator

- ***Irreducible code***: inverting code that can be decoded without any device M_i is not a prefix of M_j $\forall$ i , j

Some definitions and properties linked to information encoding and decoding:

Efficiency:

For a given alphabet A, the efficiency of a code is η given by:

$$\eta = \frac{n_{min}}{E(n)} = \frac{\min E(n)}{E(n)} = \frac{\dfrac{H}{\log_2 q}}{E(n)} = \frac{H}{E(n)\log_2 q}, \eta \in [0, 1]$$

Redundancy:

The mathematical redundancy is defined by the factor $\rho = 1 - \eta$. Redundancy can be used to increase the robustness of the coding when faced with transmission errors for the coded information (error detection and correction).

Here is a simple example: we consider a source of 4 possible messages $\{m_1, m_2, m_3, m_4\}$ of probabilities: $p_1 = 0.5$; $p_2 = 0.25$; $p_3 = p_4 = 0.125$, respectively.
Given the following two codes C_1 (simple binary codage) and C_2 (variable length code):

Messages Codes	m_1	m_2	m_3	m_4
C_1	0 0	0 1	1 0	1 1
C_2	0	1 0	1 1 0	1 1 1

For C_1 : $\eta = \frac{1.75}{2} = 0.875$ and $\rho = 1 - \eta = 0.125$.

For C_2 : $\eta = \frac{1.75}{1.75} = 1$ and $\rho = 1 - \eta = 0$.
The code C_2 is of maximum efficiency (unitary) while code C_1 is not.

Regular code:

Any given code-word is associated with only one possible message (application S→A is bijective): if $m_i \neq m_j$ then $M_i \neq M_j$.

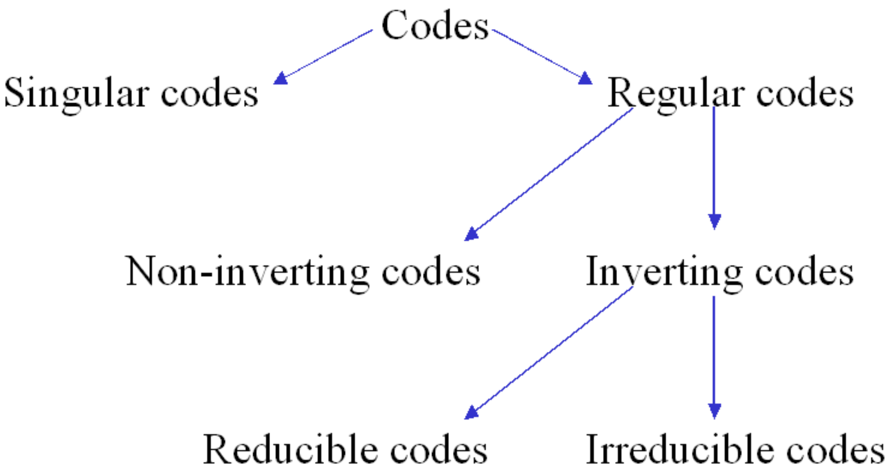
Inverting code:

The code is inverting if two distinct sets of messages $(m_{\alpha 1}, \ldots, m_{\alpha i})$ and $(m_{\beta 1}, \ldots, m_{\beta j})$ necessary lead to distinct codings (for example code of fixed length such as C_1 and codes with separator). An inverting code is then a special case of a regular code.

Irreducible code:

This is a decryptable code that can be read directly without any special device (fixed length code, separator). To do that, any code-word M_i of a message m_i must have no prefix that is another code-word M_j.

In this way, we can create a hierarchical classification to characterize a code's type:



Code examples

Regular codes / Inverting codes / Irreducible codes

Messages Proba.	m_1 0.5	m_2 0.25	m_3 0.125	m_4 0.125
C_1	1	1	0	00
C_2	0	1	11	01
C_3	1	01	001	000
C_4	1	10	100	1000

- ➢ C_1 is a regular code
- ➢ C_2 is a non-inverting code
- ➢ C_3 is an inverting and irreducible code
- ➢ C_4 is only an inverting code

Here are four codes C_1, C_2, C_3 and C_4 given as examples of the previous definitions and properties. We suppose that the four messages m_1, m_2, m_3, and m_4 are distinct.

The code C_1 is not regular: $m_1 \neq m_2$ but $C_1(m_1) = C_1(m_2)$, and also $C_1(m_3) = C_1(m_4)$.

The code C_2 is a non-inverting code: the two texts $\{m_1, m_2\}$ and $\{m_4\}$ are different, but they lead to the same code « 01 ».

The code C_3 is an inverting and irreducible code: two distinct texts made up of sequences of messages, for example $\{m_1, m_3, m_4\}$ and $\{m_1, m_2\}$ always lead to different codes and no code-word $M_i = C_3(m_i)$ is prefixed by another code-word $M_j = C_3(m_j)$

The code C_4 is an inverting code but not irreducible: two distinct texts always lead to different codes but the code-words $M_i = C_4(m_i)$ are the prefixes of all the code-words $M_j = C_4(m_j)$ once $i < j$.