



Systems and Information Theory 3 - Source Coding

3.1 Objectives

After this section you should be able to:

- Define and calculate the efficiency of a source code
- Describe the characteristics of an efficient source code
- Design a compact, instantaneous source code using the Fano and Huffman algorithms

3.2 Essential Reading

You will find this material in any elementary comms textbook, especially the following
Usher and Guy, "Information and Communication for Engineers"
Bateman, A, " Digital Communications"

3.3 Introduction

Analog signals can be sampled and converted into n -level digital signals, (and some data is initially generated in n -bit parallel format) - these n -level symbols can either be transmitted by an n -level signal, or coded into binary at the source.

We shall investigate how the type of coding used can affect the transmission rate and performance of the channel.

3.4 A Coding Example

Consider a source with an alphabet of 4 symbols:

Symbol i	P_i	I bits
A	$\frac{1}{2}$	1
B	$\frac{1}{4}$	2
C	$\frac{1}{8}$	3
D	$\frac{1}{8}$	3

Figure 3.1

The entropy is given by Equation 2.5 as

$$H = \sum_{i=1}^n P_i \log_2 \frac{1}{P_i} = 1.75 \text{ bits/symbol}$$

If, for example, the source were to transmit its symbols at 100 baud, then the information rate would be 175 bits/s.

Note that the maximum entropy H_{MAX} would be 2 bits/symbol if we had equiprobable, 4-level symbols, so the efficiency of the source is 87.5%.

Now consider *straight binary coding* of the 4-level symbols (code1):-

Symbol	P_i	Code 1
A	$\frac{1}{2}$	00
B	$\frac{1}{4}$	01
C	$\frac{1}{8}$	10
D	$\frac{1}{8}$	11

Figure 3.2

We are transmitting *two* binary symbols for each 4-level symbol, so the transmission rate is 200 bits/s (assuming that the symbols {ABCD} are generated at 100 baud.)

This however, still only conveys 175 bits/s of information, and this represents a *coding efficiency*¹ of

$$\frac{\text{Information Rate}}{\text{Transmission Rate}} = 87.5\% \quad 3.1$$

For unequal-length codes we define the coding efficiency as
$$\frac{\text{Entropy of Source}}{\text{Average Codeword Length}} \quad 3.2$$

IMPORTANT: Note the unfortunate confusion between information bits and binary symbol bits: one bit of a binary signal has 1 bit of information only when the coding efficiency is 100% and the source redundancy is 0.²

3.5 The Design of Efficient Codes

For 100% efficient coding the *average codeword length per symbol* equals the entropy of the source. Shannon proved that there is always a code or coding technique which will give an efficiency, as defined in Eqn 3.2, of 100% (See Sec 3.6)

An efficient code is a *compact* code: it can be designed using the following common-sense principle:

Symbols which occur with the highest probability should be assigned the shortest codewords.

In general, a compact code is not therefore an equal length code. The Morse Code (Appendix 3.1) is an early example of an attempt to produce an efficient code for the alphabet:³

Symbol <i>i</i>	P_i	Code 1	Code 2	Code 3	Code 4
A	1/2	00	0	0	0
B	1/4	01	1	10	10
C	1/8	10	10	110	110
D	1/8	11	11	1110	111
Av Length bits		2	1.25	1.875	1.75
Efficiency %		87.5	140	93	100

Figure 3.3 Binary Coding Examples

We note that:

Code 2 would appear to be more than 100% efficient, but it is *not uniquely decipherable* since successive codewords run-together unless we include spaces or *delimiters* (like Morse Code). This, of course, increases the codeword length and will reduce the efficiency to 100% or less.

Code 3 is a so-called *comma* code, using '0' as a *delimiter* between symbols, but is not 100% efficient because of the extra binary symbols required. The '0' is often called a "stop bit".

The inefficient binary code, **Code 1**, is *comma free* because it is equal length, and provided decoding starts at the beginning and keeps in step there is no ambiguity in the decoding.

Code 4 is *compact*, *comma free* and also *instantaneous*; each code word is unique and can be decoded immediately upon receipt.

To be an instantaneous code, no codeword can be a prefix of any other.

The average length of the code is 1.75 bits/symbol giving the required 100% coding efficiency. It is an example of a *Tree Code*.

¹ Since we are coding into equal length binary, this is the same as the source efficiency.

² This confusion would not have arisen if different names had been used for these quantities. Some authors use *binits* for binary symbols. However, the practice of referring to them as bits arose before the full implications of information theory had been appreciated.

³ Morse Code is not binary since it uses dot, dash and space, but the same principles hold for such multi-symbol codes as for binary.

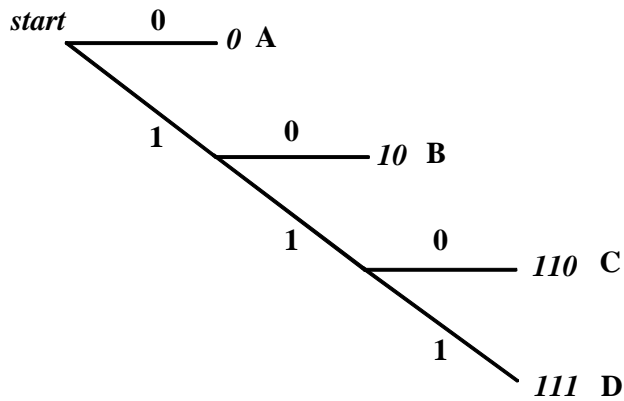


Fig 3.1 Code tree for code 4

3.6 However, ...

As we see, the binary coding of Code 1 requires a transmission rate of 200 bits/s to convey 175 bits/s of information.

Inspecting the coding table we see that $P_0 = 11/16$ and $P_1 = 5/16$. If we apply Equation 2.5 to calculate the entropy of the binary output we get 1.79 bits/symbol. The coder has apparently *increased* the information in the signal!

This embarrassment is removed if we recognise that each symbol results in a pair of binary symbols, and the two bits of each pair are not statistically *independent* of each other. We should calculate the entropy of the binary signal using conditional probabilities, as explained in section 2.7.

3.7.1 Practical source codes (1) The Fano Code

The algorithm is simple:

- 1 Divide the symbol table into two groups having, as nearly as possible, equal probability
- 2 Allocate '0' and '1' to each group
- 3 Subdivide each group into two equal probability halves
- 4 Repeat (2) and (3) until no more division is possible
- 5 Read-off the code as the string of allocated symbols

3.7.2 Practical source codes (2) The Huffman Code

The Huffman Code is more efficient for long code tables:

- 1 Arrange the symbols in descending order of probabilities
- 2 Allocate '0' and '1' to the two least probable symbols
- 3 Combine the last two symbols into a group and add their probabilities
- 4 If necessary re-order the table and repeat steps (1) to (3) until all the symbols are included
- 5 Read-off the code as the string of allocated symbols

In each case, the resulting code will be compact, but it may not be instantaneous: in that case change one or other of the '0'/'1' assignments until it is.

If the symbol probabilities are powers of $1/2$, the *Fano or Huffman* coding algorithms are guaranteed to give a 100% efficient code because the code word lengths can then exactly equal the information in each symbol. For other symbol probability distributions the coding will not be 100% efficient but can be made nearly so if the symbols are taken n at a time, and n is large. This is known as *n'th extension coding*.

3.8 Shannon's Third Theorem

Shannon's Source Coding Theorem states that

If the source symbols are coded in groups of n , then the *average length per symbol* tends to the source entropy as n tends to infinity.

That is
$$\lim_{n \rightarrow \infty} \frac{L_n}{n} = H$$

where L_n is the average length of the codewords for the n -symbol groups.

These codes are known as *dictionary codes*, where the transmitted codeword is used as a pointer to the correct entry in the table. In Huffman/Fano codes the dictionary remains unchanged, or *static*, throughout transmission. They are therefore inflexible, since the code table depends on the symbol probabilities which may change with time or type of message. You may study *dynamic dictionary* codes at a later time, where the codetable is constructed at both source and destination during transmission of the message.

3.8 Data Compression - lossless and lossy coding

The best of the codes in Sec 3.5 produces a *data compression* of 87.5% (2 bits \rightarrow 1.75 bits). Since the original message can always be reconstructed, assuming that there are no errors due to noise, they are *lossless* or *reversible* or *noiseless* compression codes. You may have used Winzip to compress your computer files - this is a good example of a lossless code.

Data compression is very important for storage and transmission of audio and video signals, since they produce very high bit rates when digitised⁴. However these signals have a lot of redundancy, and we can use *lossy/irreversible/noisy* compression to reduce the data rate to practical levels using JPEG or MPEG coding. JPEG is used for still images, and MPEG for video and the accompanying audio.

If you watch digital TV you may become aware of the disadvantage of lossy compression: when objects move quickly across the screen (eg a football), or the SNR on the channel degrades there is not enough information available to reconstruct the image quickly enough and it breaks-down into large blocks of pixels⁵, an effect known, naturally, as blocking. When an uncompressed image suffers from SNR degradation, the picture just becomes noisy or "grainy".

⁴ A simple calculation shows that an uncompressed TV picture requires a bit rate of 234 Mbit/s, and would need 872 Gbytes to store a 1 hour video. Both figures are impractical.

⁵ A pixel is a picture element, sometimes called a pel. It is the smallest element into which an image can be subdivided - an image molecule, if you like.

Appendix 3.1 Codes for the English Alphabet

Letter	Probability	Morse Code	ASCII	Huffman
A	0.082	.-	100 0001	01100
B	0.014	-...	100 0010	01111111
C	0.028	-.-.	100 0011	111111
D	0.038	-..	100 0100	01011
E	0.131	.	100 0101	101
F	0.029	..-.	100 0110	001100
G	0.020	--.	100 0111	011101
H	0.053	100 1000	1110
I	0.063	..	100 1001	1000
J	0.001	.---	100 1010	0111001110
K	0.004	-.-	100 1011	01110010
L	0.034	.-..	100 1100	01010
M	0.025	--	100 1010	001101
N	0.071	-..	100 1110	1001
O	0.080	---	100 1111	0110
P	0.020	.-.-.	101 0000	011110
Q	0.001	--.-	101 0001	0111001101
R	0.068	.-.	101 0010	1101
S	0.061	...	101 0011	1100
T	0.105	-	101 0100	0010
U	0.025	..-	101 0101	11110
V	0.009	...-	101 0110	0111000
W	0.015	.-.-	101 0111	001110
X	0.002	-.-.-	101 1000	0111001100
Y	0.020	-.--	101 1001	001111
Z	0.001	--..	101 1010	0111001111

Letter	Probability	Morse Code	ASCII	Huffman
E	0.131	.	100 0101	101
T	0.105	-	101 0100	0010
A	0.082	.-	100 0001	01100
O	0.080	---	100 1111	0110
N	0.071	-..	100 1110	1001
R	0.068	.-.	101 0010	1101
I	0.063	..	100 1001	1000
S	0.061	...	101 0011	1100
H	0.053	100 1000	1110
D	0.038	-..	100 0100	01011
L	0.034	.-..	100 1100	01010
F	0.029	..-.	100 0110	001100
C	0.028	-.-.	100 0011	111111
M	0.025	--	100 1010	001101
U	0.025	..-	101 0101	11110
G	0.020	--.	100 0111	011101
P	0.020	.-.-.	101 0000	011110
Y	0.020	-.--	101 1001	001111
W	0.015	.-.-	101 0111	001110
B	0.014	-...	100 0010	01111111
V	0.009	...-	101 0110	0111000
K	0.004	-.-	100 1011	01110010
X	0.002	-.-.-	101 1000	0111001100
J	0.001	.---	100 1010	0111001110
Q	0.001	--.-	101 0001	0111001101
Z	0.001	--..	101 1010	0111001111

