CHAPTER 8 INFORMATION THEORY AND HUFFMAN CODING

Learning Outcomes

INFORMATION THEORY AND HUFFMAN CODING

- **Understand basic concepts of information theory**
  - ❖ Explain the following terms: information content of a symbol, average information contents of symbols and entropy.
  - ❖ Determine the information content of a symbol, average information contents of symbols and entropy for discrete sources.
- **Apply knowledge of source coding**
  - ❖ Explain the need for redundancy removal and its benefits.
  - ❖ Explain the techniques of Fixed Length Coding and Variable length coding.
  - ❖ Identify the properties of a useful code.
  - ❖ Design the Huffman Code for discrete sources.
  - ❖ Evaluate the merit of source codes using the concepts of compression ratio and code efficiency.

8.1. INTRODUCTION TO INFORMATION THEORY

The information content of a message depends on the probability of the event associated with the message. The lower the probability of occurrence of the message, the higher is the information content it carries.

In a digital communication system, the source is generating symbols to form messages. It is therefore more convenient to consider the information content on per symbol basis rather than per message.

In defining the information content of symbols, we need to keep two factors in mind:

- The instantaneous flow of information fluctuates widely due to randomness of symbol transmission. Hence we take the average information content of symbols in a long message.
- The statistical dependence of symbols in a message sequence alters the average information content of the symbols.

To simplify our discussion, we assume symbols are emitted independently by the discrete source.

8.1.1 Average Information Content of symbols (ENTROPY)

Suppose a source emits one of M possible symbols, $S_1, S_2, S_3, \ldots S_M$. Let $P_1, P_2, P_3, \ldots P_M$ be the probabilities of occurrence of the M symbols, respectively.

Then the information content, I of the symbol S_i is defined as:

$$I(S_i) = \log_2 (1/P_i) \text{ bits} \quad (8.1)$$

where P_i is the probability of occurrence of the symbol, S_i.

To obtain the average information content of the M symbols, consider a long message of N symbols (N being very large). On the average, the ith symbol, S_i will occur NP_i times.

In this message, the information content, I contributed by the symbol S_i is then

$$I(S_i) = NP_i \log_2 (1/P_i) \text{ bits} \quad (8.2)$$

and the total information content of the message, due to the contribution of all the symbols ($S_1, S_2, S_3, \ldots S_M$) is

$$I\ (\text{total}) = \sum_{i=1}^{M} NP_i \log_2 (1/P_i) \text{ bits} \qquad (8.3)$$

The average information content per symbol in this message (N symbols long) is hence

$$I\ (\text{total})/\ N = \sum_{i=1}^{M} P_i \log_2 (1/P_i) = - \sum_{i=1}^{M} P_i \log_2 P_i \text{ bits/symbol} \qquad (8.4)$$

The average information content per symbol is known as ENTROPY, H.

The unit for H is average bits per event. The unit bit, here, is a measure of information content and is not to be confused with the term "bit", meaning "binary digit".

The average rate of information going through a channel is:

Average rate of information = $H(X) \times r_s$.

Image Entropy

The image entropy specifies the uncertainty in the image values.
Measures the averaged amount of information required to encode the image values.

Example 8.1

Find the entropy of a source that emits one of these symbols A, B and C in a statistically independent sequence with probabilities 1/2, 1/4 and 1/4, respectively.

Solution

$P(A) = 1/2$; $P(B) = 1/4$; $P(C) = 1/4$

Information contents of A, B and C are

$\log_2 (1/P(A)) = \log_2 2 = 1$ bit

$\log_2 (1/P(B)) = \log_2 (1/P(C)) = \log_2 4 = 2$ bits

Entropy $H = P(A) \log_2 (1/P(A)) + P(B) \log_2 (1/P(B)) + P(C) \log_2 (1/P(C))$
$= 1/2 \times 1 + 1/4 \times 2 + 1/4 \times 2$
= 1.5 bits/symbol

Example 8.2

A discrete source emits one of five symbols once every millisecond. The symbol probabilities are $P_1 = 1/2$; $P_2 = 1/4$; $P_3 = 1/8$; $P_4 = 1/16$; $P_5 = 1/16$. Find the source entropy and information rate.

<u>Solution</u>

Symbol rate $r_s = 1000$ symbols/sec

Entropy $H = \sum_{i=1}^{5} P_i \log_2 1/P_i$

$H = 1/2 \log_2 2 + 1/4 \log_2 4 + 1/8 \log_2 8 + 1/16 \log_2 16 + 1/16 \log_2 16$

$H = \underline{1.875 \text{ bits/symbol}}$

Information rate $= Hr_s = \underline{1875 \text{ bits/sec}}$

Example 8.3

Average Information Content in the English Language

a. Calculate the average information in bits/character for the English language, assuming that each of the 26 characters in the alphabet occurs with equal likelihood. Neglect spaces and punctuation.

b. Since the alphabetic characters do not appear with equal frequency in the English language (or any other language), the answer to part (a) will represent an upper bound on average information content per character. Repeat part (a) under the assumption that the alphabetic characters occur with the following probabilities:

   P = 0.10: for the letters, a, e, o, t
   P = 0.07: for the letters, h, i, n, r, s
   P = 0.02: for the letters, c, d, f, l, m, p, u, y
   P = 0.01: for the letters, b, g, j, k, q, v, w, x, z

<u>Solution</u>

(a) Entropy $H = -\sum_{i=1}^{26} 1/26 \log_2 (1/26)$

$= \underline{4.7 \text{ bits/character}}$

(b) Entropy H = -(4 x 0.1 $\log_2$ 0.1 + 5 x 0.07 $\log_2$ 0.07 + 8 x 0.02 $\log_2$ 0.02 + 9 x 0.01 $\log_2$ 0.01)

= 4.17 bits/character

Note: If fixed length coding is used, 5 bits are needed.

8.2 SOURCE CODING

8.2.1 Redundancy-Reducing Coding

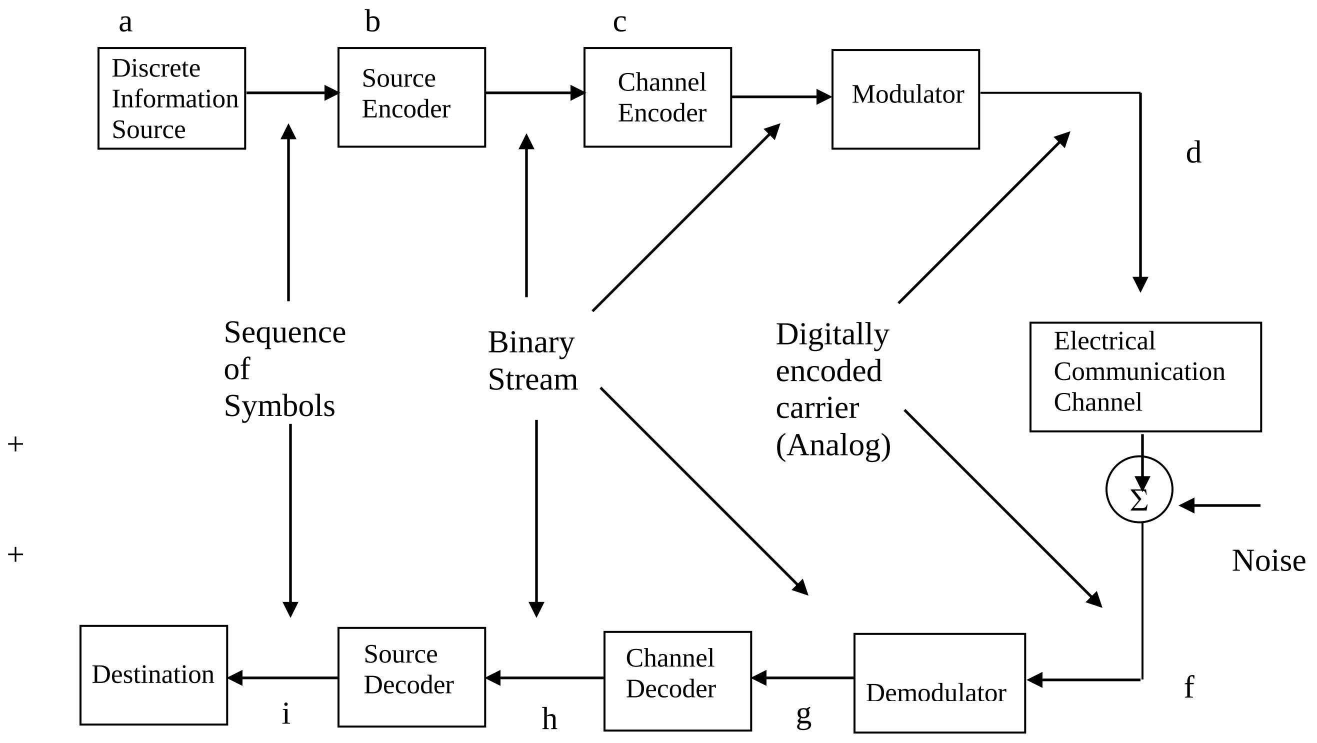

Figure 8.1 Block diagram of a Digital Communication System

Figure 8.1 shows the block diagram of a typical digital communication systems. Our emphasis will be on the source decoder for this chapter. Source coding deals with efficient descriptions of information sources. The advantage of source coding is reduction in bandwidth and/or energy per bit required to represent the source, at the expense of computation and memory. We will concentrate on the coding of discrete sources.

Such coding requires the selection of an efficient (usually) binary representation of the source. Often it also requires substitution of one binary representation of the source symbols with an alternative representation.

In fixed-length codes each symbol is represented by codewords having the same number of bits. Usually, when all symbols are equally likely, the code should be of fixed length.

One technique in source coding is to use variable-length coding. In variable-length codes, longer codewords are assigned to symbols that do not occur as often. Intuitively, the length of codeword is inversely related to the probability of that symbol. As a symbol with high probability, contains little information and should not be assigned much of the system resources. What we receive in exchange for using the occasional longer codewords is more efficient storage or transmission of the source. Data compression codes are often variable-length codes. The best known variable-length code is the Morse code where code assignment reflects the relative frequency of the symbols.

A significant amount of data compression can be realised when there is a wide difference in the probabilities of the symbols. To achieve this compression there must also be a sufficient large number of symbols. Sometimes, in order to have a large enough set of symbols, we form a new set of symbols derived from the original set called the extension code.

8.2.2 Properties of Codes

There are properties that a code must satisfy for it to be useful. We will consider a three-symbol alphabet with the given probability assignment, shown below. Listed with the input alphabet are six binary code assignments. Scan these for a moment and try to determine which codes are practical.

X_I	$P(X_i)$
a	0.73
b	0.25
c	0.02

Table 8.1 Three-symbol alphabet

Symbol	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6
a	00	00	0	1	1	1
b	00	01	1	10	00	01
c	11	10	11	100	01	11

Table 8.2 Six binary code assignments

Uniquely Decodeable Property

Uniquely decodeable codes are those that allow us to invert the mapping to the original symbol alphabet. Obviously, code 1 is not uniquely decodeable because the symbol a and b are assigned the same binary sequence. Thus first requirement of a useful code is that each symbol be assigned a unique binary sequence.

By this condition, all other codes appear satisfactory until we examine codes 3 and 6 carefully. These codes have unique binary codewords assigned to each symbol. The problem occurs when these codewords are strung together. For instance, try to decode the binary pattern 1 0 1 1 1

In code 3, is it b, a, b, b, b or b, a, b, c or b, a, c, b?

Code 6 gives similar difficulties. Because of this these codes are not uniquely decodeable even though the individual symbols have unique codewords.

Prefix-Free Property

A condition to assure unique decodeable code is that no codeword be the prefix of any other codeword. Codes that satisfy this condition are called prefix-free codes.

Note that code 4 is not prefix-free but is uniquely decodeable. Prefix-free codes also have the property that they are instantaneously decodeable. Code 4 is not instantaneously decodeable.

An instantaneously decodeable code is one for which the boundary of the present codeword can be identified by the end of the present codeword rather than by the beginning of the next codeword.

For instance, in transmitting the symbol b with the binary sequence 1 0 in code 4, the receiver cannot determine if this is the whole codeword for symbol b or the partial codeword for symbol c.

8.2.3 Code Length and Source Entropy

The average bit length achieved by a given code is denoted by $\bar{n}$. This average length is computed as the sum of the binary code lengths n_i weighted by the probability of that code symbol $P(X_i)$.

$$\bar{n} = \sum_{i=1}^{M} n_i P(X_i) \qquad (8.5)$$

Entropy represents the minimum possible average bit length achievable by a variable-length code. A number of reasons prevent an actual code from achieving the entropy bound of the input alphabet. These include uncertainty in probability assignment and buffering constraints.

8.2.4 Huffman Code

The Huffman Code is prefix-free variable-length code which can achieve the shortest average code length $\bar{n}$ for a given input alphabet. This $\bar{n}$ may be significantly greater than the entropy of the source alphabet, due to the alphabet, not to the coding technique.

Often the alphabet can be modified to form an extension code, and the same coding technique is then reapplied to achieve better compression performance.

Compression performance is measured by the **compression ratio**. This measure is equal to the ratio of the average number of bits per symbol before compression to the average number of bits per symbol after compression.

Code efficiency is another measure of compression performance. This measure is equal to the ratio of the source entropy to the average number of bits per symbol after compression.

The Huffman code is generated as part of a tree-forming process.

- The process starts by listing the input symbols of the alphabet, along with their probabilities (or relative frequencies), in a decreasing order of occurrence. These tabular entries correspond to the branch ends of a tree.

- Each branch is assigned a branch weight equal to the probability of that branch. The two entries with the lowest probabilities are merged (at a branch node) to form a new branch with their composite probability.

- After every merging, the new branch and the remaining branches are reordered (if necessary) to assure that the reduced table preserves the descending probability of occurrence. We call this reordering bubbling.

- After forming the tree, each branch is labelled with a binary 1/0 decision to distinguish the two branches. The labelling is arbitrary, but if "1" is to be sent as often as possible, we label the branch going up with a "1" and the branch going down with a "0".

- After labelling the branch nodes we trace the tree path from the base of the tree (far right) to each output branch (far left). The path contains the binary sequence to reach that branch. This binary sequence is the codeword for that symbol.

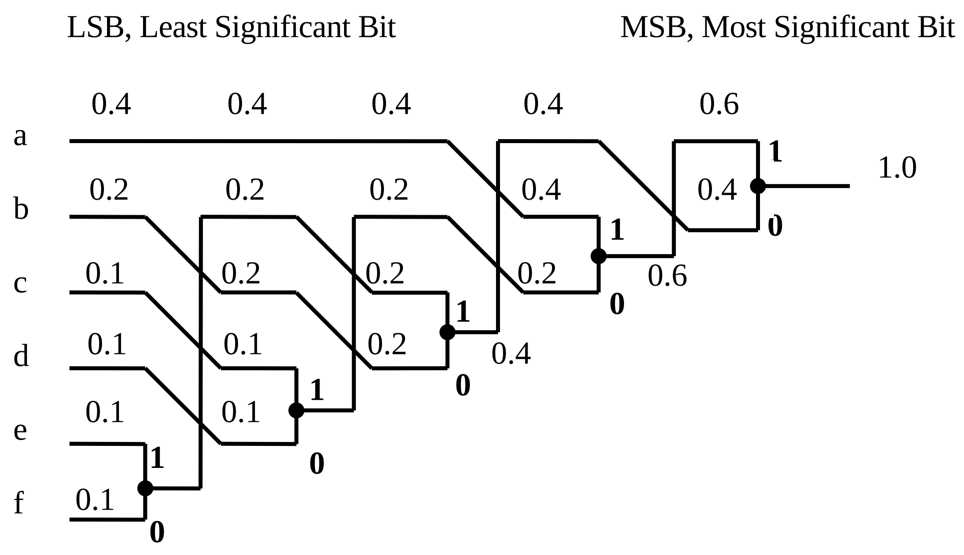

Figure 8.2 Huffman coding tree for a six-symbol set

Table 8.3 below lists the resulting codewords.

X_i	$P(X_i)$	Code	n_i	$n_iP(X_i)$
a	0.4	11	2	0.8
b	0.2	00	2	0.4
c	0.1	101	3	0.3
d	0.1	100	3	0.3
e	0.1	011	3	0.3
f	0.1	010	3	0.3
				$\bar{n}$ =2.4

where i = 1, 2, ..., 6

Table 8.3 Huffman code for a six-symbol set

$\bar{n}$, for this alphabet is 2.4 bits per character. That means on the average, 240 bits will have to be moved through the communication channel when transmitting 100 input symbols. **Redundancy of the Huffman code is 0.08 bits/symbol.**

For comparison, a fixed-length code required to span the six-character input alphabet would be of length 3 bits, and the entropy of the input alphabet, using Equation (8.4) is 2.32 bits. Thus this code offers a compression ratio of 1.25 (3.0/2.4) and achieves 96.7% (2.32/2.40) of the possible compression ratio.

Some observations on Huffman Coding

Consider the example given in Figure 8.2; the following observations can be made.

- There are five nodes in this example and with two possible assignments of "0" and "1" at each branch; there are a total of 2^5 = 32 possible combination of assignments that is 32 possible Huffman solution. Hence, the code we have generated is not necessarily unique.

- Error propagation. This setback of Huffman coding is true of any variable-length code.