# 1 Source Coding

> Not everything that can be counted counts,
> and not everything that counts can be counted.
>
> —Albert Einstein (1879–1955)

## 1.1 INTRODUCTION TO INFORMATION THEORY

Today we live in the information age. The internet has become an integral part of our lives, making this the third planet from the sun a global village. People talking over cellular phones is a common sight, sometimes even in cinema theaters. Movies can be rented in the form of a DVD disk. E-mail addresses and web addresses are common on business cards. Many people prefer to send e-mails and e-cards to their friends rather than the regular snail mail. Stock quotes can be checked over the mobile phone.

Information has become the key to success (it has always been the key to success, but in today's world it is *the* key). And behind all this exchange of information lies the tiny 1's and 0's (the omnipresent bits) that hold information merely by the way they sit next to one another. Yet the information age that we live in today is traced primarily to a seminal paper published in 1948 that laid the foundation of the wonderful field of **Information Theory**—a theory that was initiated by one man—the American Electrical Engineer Claude E. Shannon, whose ideas appeared in the article "The Mathematical Theory of Communication" in the *Bell System Technical Journal* (1948). In its broadest sense, information is interpreted to include the messages occurring in any of the standard communication media, such as telegraphy, telephony, radio,

television, and the signals involved in electronic computers, servo–mechanism systems, and other data–processing devices. The theory is even applicable to the signals appearing in the nerve networks of humans and other animals.

The chief concern of information theory is to discover mathematical laws governing the systems, designed to communicate or manipulate information. It sets up quantitative measures of information and the capacity of various systems to transmit, store, and otherwise process information. Some of the problems treated, are related in finding the best methods of using various available communication systems and the best methods for separating the wanted information, or signal, from the extraneous information, or noise. Another problem is the setting of the upper bounds on what it is possible to achieve with a given information–carrying medium (often called an information channel). While the results are chiefly of interest to communication engineers, some of the concepts have been adopted and found useful in such fields as psychology and linguistics. The notion of mutual information has also found applications in population based gene mapping, whose aim is to find DNA regions (genotypes) responsible for particular traits (phenotypes).

The boundaries of information theory are quite fuzzy. The theory overlaps heavily with communication theory but is more oriented toward the fundamental limitations on the processing and the communication of information and less oriented toward the detailed operation of the devices employed.

In this chapter, we shall first develop an intuitive understanding of information. It will be followed by the mathematical models of information sources, and a quantitative measure of the information emitted by a source. We shall then state and prove the source coding theorem. Having developed the necessary mathematical framework, we shall look at four source coding techniques—the **Huffman encoding**, the **Shannon–Fano–Elias encoding**, the **Arithmetic encoding** and the **Lempel–Ziv encoding**. This chapter will then discuss the basics of the **run length encoding**. The concept of the **rate distortion function** and the **optimum quantizer** will then be introduced. To account for random variables that are dependent, we will study the entropy rate of a stochastic process. The chapter concludes with an introduction to image compression, one of the important application areas of source coding. In particular, the **JPEG** (Joint Photographic Experts Group) standard will be discussed in brief.

## 1.2 UNCERTAINTY AND INFORMATION

Any information source produces an output that is random in nature. If the source output had no randomness *i.e.*, the output were known exactly, there would be no need to transmit it. There exist both **analog** and **discrete** information sources. Actually, we live in an analog world, and most sources are analog sources, for example, speech, temperature fluctuations, etc. The discrete sources are man–made sources, for example, a source (say, a man) that generates a sequence of letters from a finite alphabet (typing his e–mail).

Before we go on to develop a mathematical measure of information, let us develop an intuitive feel for it. Read the following sentences:



(A) Tomorrow, the sun will rise from the east.
(B) The phone will ring in the next one hour.
(C) It will snow in Delhi next winter.

The three sentences carry different amounts of information. In fact, the first sentence hardly carries any information. It is a sure-shot thing. Everybody knows that the sun rises from the east and the probability of this happening again is *almost* unity (*"Making predictions is risky, especially when it involves the future."*–N. Bohr). Sentence (B) appears to carry more information than sentence (A). The phone may ring, or it may not. There is a finite probability that the phone will ring in the next one hour (unless the maintenance people are at work again). The last sentence probably made you read it over twice. This is because it has never snowed in Delhi, and the probability of a snowfall is very low. It is interesting to note that the amount of information carried by the sentences listed above has something to do with the probability of occurrence of the events stated in the sentences. And we observe an inverse relationship. Sentence (A), which talks about an event which has a probability of occurrence very close to 1 carries almost no information. Sentence (C), which has a very low probability of occurrence, appears to carry a lot of information (made us read it twice to be sure we got the information right). The other interesting thing to note is that, the length of the sentence has nothing to do with the amount of information it conveys. In fact, sentence (A) is the longest of the three sentences but carries the minimum information.

We will now develop a mathematical measure of information.

**Definition 1.1** Consider a discrete random variable X with the possible outcomes $x_i$, $i = 1, 2, ..., n$. The **Self-Information** of the event $X = x_i$ is defined as

$$I(x_i) = \log\left(\frac{1}{P(x_i)}\right) = -\log P(x_i) . \qquad (1.1)$$

We note that a high probability event conveys less information than a low probability event. For an event with $P(x_i) = 1$, $I(x_i) = 0$. Since a lower probability implies a higher degree of uncertainty (and *vice-versa*), a random variable with a higher degree of uncertainty contains more information. We will use this correlation between uncertainty and information for physical interpretation throughout this chapter.

The units of $I(x_i)$ are determined by the base of the logarithm, which is usually selected as 2 or $e$. When the base is 2, the units are in **bits** and when the base is $e$, the units are in **nats** (natural units). Since $0 \le P(x_i) \le 1$, $I(x_i) \ge 0$, *i.e.*, the self-information is *non–negative*. The following two examples illustrate why a logarithmic measure of information is appropriate.

**Example 1.1** Consider a binary source which tosses a fair coin and outputs a 1 if a head appears and a 0 if a tail appears. For this source, $P(1) = P(0) = 0.5$. The information content of each output from the source is

$$I(x_i) = -\log_2 P(x_i) = -\log_2(0.5) = 1 \text{ bit.} \quad (1.2)$$

Indeed, we have to use only one bit to represent the output from this binary source (say, we use a 1 to represent H and a 0 to represent T). Now, suppose the successive outputs from this binary source are statistically independent, i.e., the source is memoryless. Consider a block of $m$ binary digits. There are $2^m$ possible $m$–bit blocks, each of which is equally probable with probability $2^{-m}$. The self-information of an $m$–bit block is

$$I(x_i) = -\log_2 P(x_i) = -\log_2 2^{-m} = m \text{ bits.} \quad (1.3)$$

Again, we observe that, we indeed need $m$ bits to represent the possible $m$–bit blocks. Thus, this logarithmic measure of information possesses the desired additive property when a number of source outputs is considered as a block.

**Example 1.2** Consider a discrete, memoryless source (source C), that outputs *two* bits at a time. This source comprises two binary sources (sources A and B) as mentioned in Example 1.1, each of the sources contributing one bit. The two binary sources within the source C are independent. Intuitively, the information content of the aggregate source (source C) should be the *sum* of the information contained in the outputs of the two independent sources that constitute this source C. Let us look at the information content of the outputs of source C. There are four possible outcomes {00, 01, 10, 11}, each with a probability $P(C) = P(A)P(B) = (0.5)(0.5) = 0.25$, because of the source A and B being independent. The information content of each output from the source C is

$$I(C) = -\log_2 P(x_i) = -\log_2(0.25) = 2 \text{ bits.} \quad (1.4)$$

We have to use two bits to represent the output from this combined binary source. Thus, the logarithmic measure of the information possesses the desired additive property for independent events.

Next, consider *two* discrete random variables X and Y with possible outcomes $x_i$, $i = 1, 2, ..., n$. and $y_j$, $j = 1, 2, ..., m$ respectively. Suppose we observe some outcome $Y = y_j$ and we want to determine the amount of information this event provides about the event $X = x_i$, $i = 1, 2, ..., n$, i.e., we want to mathematically represent the mutual information. We note the two extreme cases:

(i) X and Y are independent, in which case the occurrence of $Y = y_j$ provides no information about $X = x_i$.

(ii) X and Y are fully dependent events, in which case the occurrence of $Y = y_j$ determines the occurrence of the event $X = x_i$.

A suitable measure that satisfies these conditions is the logarithm of the ratio of the conditional probability



$$P(X = x_i \mid Y = y_j) = P(x_i \mid y_j), \quad (1.5)$$

divided by the probability

$$P(X = x_i) = P(x_i). \quad (1.6)$$

**Definition 1.2** The **Mutual Information** $I(x_i; y_j)$ between $x_i$ and $y_j$ is defined as

$$I(x_i; y_j) = \log\left(\frac{P(x_i \mid y_j)}{P(x_i)}\right). \quad (1.7)$$

As before, the units of $I(x)$ are determined by the base of the logarithm, which is usually selected as 2 or $e$. When the base is 2 the units are in bits. Note that

$$\frac{P(x_i|y_j)}{P(x_i)} = \frac{P(x_i|y_j)P(y_i)}{P(x_i)P(y_i)} = \frac{P(x_i, y_j)}{P(x_i)P(y_j)} = \frac{P(y_j|x_i)}{P(y_j)}. \quad (1.8)$$

Therefore,

$$I(x_i; y_j) = \log\left(\frac{P(x_i|y_j)}{P(x_i)}\right) = \log\left(\frac{P(y_j|x_i)}{P(y_j)}\right) = I(y_j; x_i). \quad (1.9)$$

The physical interpretation of $I(x_i; y_j) = I(y_j; x_i)$ is as follows: The information provided by the occurrence of the event $Y = y_j$ about the event $X = x_i$ is identical to the information provided by the occurrence of the event $X = x_i$ about the event $Y = y_j$.

Let us now verify the two extreme cases:

(i) When the random variables X and Y are statistically independent, $P(x_i \mid y_j) = P(x_i)$, which leads to $I(x_i; y_j) = 0$.

(ii) When the occurrence of $Y = y_j$ uniquely determines the occurrence of the event $X = x_i$, $P(x_i \mid y_j) = 1$, and the mutual information becomes

$$I(x_i; y_j) = \log\left(\frac{1}{P(x_i)}\right) = -\log P(x_i). \quad (1.10)$$

This is the self-information of the event $X = x_i$.

Thus, the logarithmic definition of mutual information confirms our intuition.

**Example 1.3** Consider a **binary symmetric channel (BSC)** as shown in Fig. 1.1. A binary channel can be visualised as one that transports 1's and 0's from the transmitter ($T_x$) to the receiver ($R_x$). It makes an error occasionally, with a probability $p$. A BSC channel flips a 1 to 0 and *vice–versa* with equal probability. Let X and Y be the binary random variables that represent the input and output of this BSC. Let the input symbols be equally likely,

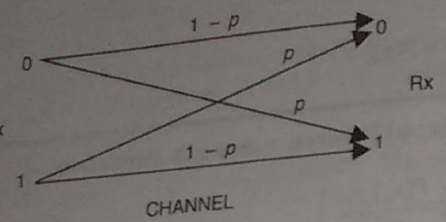


Fig. 1.1 A binary symmetric channel.

and the output symbols depend upon the input according to the channel transition probabilities are given below:

$$P(Y=0 \mid X=0) = 1-p$$
$$P(Y=0 \mid X=1) = p$$
$$P(Y=1 \mid X=1) = 1-p$$
$$P(Y=1 \mid X=0) = p$$

It simply implies that the probability of a bit getting flipped (in error) when transmitted over this BSC is $p$. From the channel transition probabilities we have

$$P(Y=0) = P(X=0).P(Y=0 \mid X=0) + P(X=1).P(Y=0 \mid X=1)$$
$$= 0.5(1-p) + 0.5(p) = 0.5, \text{ and,}$$
$$P(Y=1) = P(X=0).P(Y=1 \mid X=0) + P(X=1).P(Y=1 \mid X=1)$$
$$= 0.5(p) + 0.5(1-p) = 0.5$$

Suppose that we are at the receiver end and we want to comment about what was transmitted at the transmitter, having observed what was received at the receiver. The mutual information about the occurrence of the event $X = 0$ given that $Y = 0$ is observed is

$$I(x_0;y_0) = I(0;0) = \log_2\left(\frac{P(Y=0|X=0)}{P(Y=0)}\right) = \log_2\left(\frac{1-p}{0.5}\right) = \log_2 2(1-p).$$

Similarly,

$$I(x_1;y_0) = I(1;0) = \log_2\left(\frac{P(Y=0|X=1)}{P(Y=0)}\right) = \log_2\left(\frac{p}{0.5}\right) = \log_2 2p.$$

Let us consider some specific cases. Suppose $p = 0$, *i.e.*, it is an ideal channel (noiseless). In that case

$$I(x_0; y_0) = I(0; 0) = \log_2 2(1-p) = 1 \text{ bit.}$$

Hence having observed the output we can determine what was transmitted with certainty. Recall that the self-information about the event $X = x_0$ was 1 bit. However, if $p = 0.5$, we obtain

$$I(x_0; y_0) = I(0; 0) = \log_2 2(1-p) = \log_2 2(0.5) = 0.$$

This implies that having observed the output, we have *no* information about what was transmitted. Thus, it is a useless channel. For such a channel, there is no point in observing the received symbol and trying to make a guess as to what was sent. Instead we can as well toss a fair coin at the receiver in order to estimate what was sent.

Suppose we have a channel where $p = 0.1$. For this channel,

$$I(x_0; y_0) = I(0; 0) = \log_2 2(1-p) = \log_2 2(0.9) = 0.848 \text{ bits.}$$



**Example 1.4** Let $X$ and $Y$ be binary random variables that represent the input and output of a binary channel shown in Fig. 1.2. Let the input symbols be equally likely, and the output symbols depend upon the input according to the channel transition probabilities:

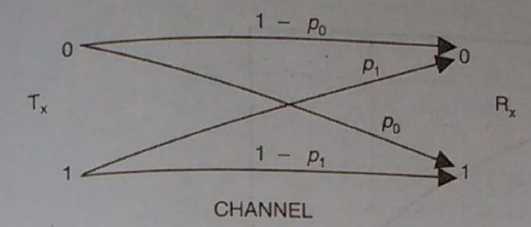


Fig. 1.2 A binary channel with asymmetric probabilities.

$$P(Y=0 \mid X=0) = 1-p_0$$
$$P(Y=0 \mid X=1) = p_1$$
$$P(Y=1 \mid X=1) = 1-p_1$$
$$P(Y=1 \mid X=0) = p_0$$

From the channel transition probabilities we have

$$P(Y=0) = P(X=0).P(Y=0 \mid X=0) + P(X=1).P(Y=0 \mid X=1)$$
$$= 0.5(1-p_0) + 0.5(p_1) = 0.5(1-p_0+p_1), \text{ and,}$$
$$P(Y=1) = P(X=0).P(Y=1 \mid X=0) + P(X=1).P(Y=1 \mid X=1)$$
$$= 0.5(p_0) + 0.5(1-p_1) = 0.5(1-p_1+p_0).$$

Suppose, that we are at the receiver end and we want to comment about what was transmitted at the transmitter, having observed what is received at the receiver. The mutual information about the occurrence of the event $X = 0$ given that $Y = 0$ is observed is

$$I(x_0;y_0) = I(0;0) = \log_2\left(\frac{P(Y=0|X=0)}{P(Y=0)}\right) = \log_2\left(\frac{1-p_0}{0.5(1-p_0+p_1)}\right) = \log_2\left(\frac{2(1-p_0)}{1-p_0+p_1}\right).$$

Similarly,

$$I(x_1;y_0) = I(1;0) = \log_2\left(\frac{P(Y=0|X=1)}{P(Y=0)}\right) = \log_2\left(\frac{2p_1}{1-p_0+p_1}\right).$$

**Definition 1.3** The **Conditional Self-Information** of the event $X = x_i$ given $Y = y_j$ is defined as

$$I(x_i|y_j) = \log\left(\frac{1}{p(x_i|y_j)}\right) = -\log\ p(x_i|y_j). \tag{1.11}$$

Thus, we may write

$$I(x_i; y_j) = I(x_i) - I(x_i \mid y_j). \tag{1.12}$$

The conditional self-information can be interpreted as the self-information about the event $X = x_i$ having observed the event $Y = y_j$. Recall that both $I(x_i) \geq 0$ and $I(x_i \mid y_j) \geq 0$. Therefore, $I(x_i; y_j) < 0$ when $I(x_i) < I(x_i \mid y_j)$ and $I(x_i; y_j) > 0$ when $I(x_i) > I(x_i \mid y_j)$. Hence, mutual information can be positive, negative or zero.

**Example 1.5** Consider the BSC discussed in Example 1.3. The plot of the mutual information $I(x_0; y_0)$ versus the probability of error, $p$ is given in Fig. 1.3.

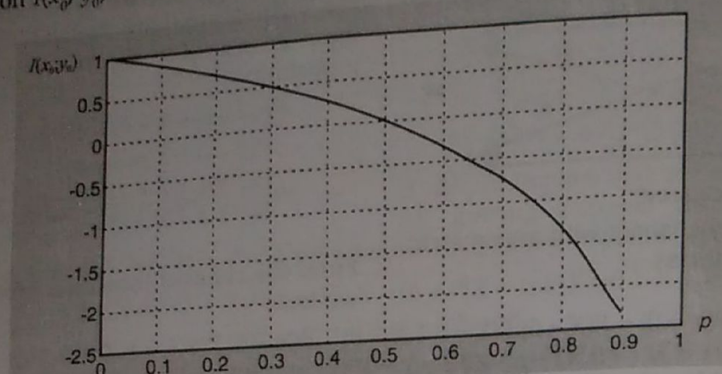


**Fig. 1.3** The plot of the mutual information $I(x_0; y_0)$ versus the probability of error, $p$.

It can be seen from the figure that $I(x_0; y_0)$ is negative for $p > 0.5$. The physical interpretation is as follows. A negative mutual information implies that having observed $Y = y_0$, we must *avoid* choosing $X = x_0$ as the transmitted bit.

For $p = 0.1$,

$$I(x_0; y_1) = I(0; 1) = \log_2(p) = \log_2(0.1) = -2.322 \text{ bits.}$$

This shows that the mutual information between the events $X = x_0$ and $Y = y_1$ is negative for $p = 0.1$. For the extreme case of $p = 1$, we have

$$I(x_0; y_1) = I(0; 1) = \log_2(p) = \log_2(1) = 1 \text{ bit.}$$

The channel always changes a 0 to a 1 and *vice-versa* (since $p = 1$). This implies that if $y_1$ is observed at the receiver, it can be concluded that $x_0$ was actually transmitted. This is actually a useful channel with a 100 % bit error rate. We just flip the received bit.

## 1.3 AVERAGE MUTUAL INFORMATION AND ENTROPY

So far we have studied the mutual information associated with a pair of events $x_i$ and $y_j$ which are the possible outcomes of the two random variables $X$ and $Y$. We now want to find out the average mutual information between the two random variables. This can be obtained simply by weighting $I(x_i; y_j)$ by the probability of occurrence of the joint event and summing over all possible joint events.

**Definition 1.4** The **Average Mutual Information** between two random variables $X$ and $Y$ is given by



$$I(X;Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j) I(x_i; y_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)},$$

$$= \sum_{j=1}^{n}\sum_{j=1}^{m} P(x_i) P(y_j \mid x_i) \log \frac{P(y_j \mid x_i)}{P(y_j)},$$

$$= \sum_{j=1}^{n}\sum_{i=1}^{m} P(y_j) P(x_i \mid y_j) \log \frac{P(x_i \mid y_j)}{P(x_i)}. \quad (1.13)$$

For the case when $X$ and $Y$ are statistically independent, $I(X; Y) = 0$, *i.e.*, there is no average mutual information between $X$ and $Y$. An important property of the average mutual information is that $I(X; Y) \geq 0$, with equality, if and only if $X$ and $Y$ are statistically independent.

**Definition 1.5** The **Average Self-Information** of a random variable $X$ is defined as

$$H(X) = \sum_{i=1}^{n} P(x_i)\, I(x_i) = -\sum_{i=1}^{n} P(x_i) \log P(x_i). \quad (1.14)$$

When $X$ represents the alphabet of possible output letters from a source, $H(X)$ represents the average information per source letter. In this case $H(X)$ is called the **Entropy**. The entropy of $X$ can be interpreted as the expected value of $\log\left(\frac{1}{P(X)}\right)$. The term entropy has been borrowed from statistical mechanics, where it is used to denote the level of disorder in a system. It is interesting to see that the Chinese character for entropy looks like 熵.

We observe that since $0 \leq P(x_i) \leq 1$, $\log\left(\frac{1}{P(X)}\right) \geq 0$. Hence, $H(X) \geq 0$.

**Example 1.6** Consider a discrete binary source that emits a sequence of statistically independent symbols. The output is either a 0 with probability $p$ or a 1 with a probability $1 - p$. The entropy of this binary source is

$$H(X) = -\sum_{i=0}^{1} P(x_i) \log P(x_i) = -p \log_2 (p) - (1-p) \log_2 (1-p). \quad (1.15)$$

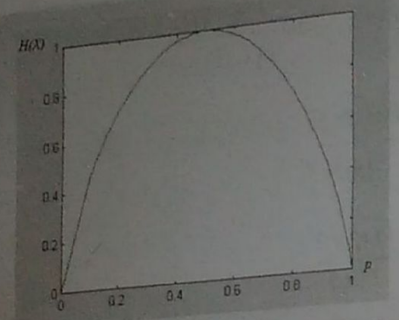


Fig. 1.4 The entropy function, $H(X) = -p \log_2(p) - (1-p) \log_2(1-p)$.

The plot of the *binary entropy function* versus $p$ is given in Fig. 1.4.

We observe from the figure that the value of the binary entropy function reaches its maximum value for $p = 0.5$, *i.e.*, when both 1 and 0 are equally likely. In general it can be shown that the entropy of a discrete source is maximum when the letters from the source are equally probable.

**Example 1.7** Consider the English language with alphabet {A, B, ..., Z}. If every letter occurred with the same probability and was independent from the other letters, then the entropy per letter would be $\log_2 26 = 4.70$. This is the absolute upperbound. However, we know that all letters do not appear with equal probability. If we take into consideration the probabilities of occurrences of different alphabets (normalised letter frequency), the entropy per letter, $H_L$, would be

$$H_L \leq H(X) \approx 4.14 \text{ bits.} \tag{1.16}$$

This is still an over-estimate because the letters are not independent. For example, Q is always followed by U and the **bigram** TH is likely to occur frequently. Thus a better estimate of the entropy per letter could be possibly obtained by looking at the bigrams. If $X^2$ denotes the random variable of bigrams in the English language, the upperbound on $H_L$ can be refined as

$$H_L \leq \frac{H(X^2)}{2} \approx 3.56 \text{ bits.} \tag{1.17}$$

This logic can be extended to ***n*–grams**. Thus the entropy of the language can be defined as

$$H_L = \lim_{n \to \infty} \frac{H(X^n)}{n}. \tag{1.18}$$

Even though the exact value of $H_L$ may be difficult to determine, statistical investigations show that for the English language

$$1 \leq H_L \leq 1.5 \text{ bits.} \tag{1.19}$$

So for each letter in the English text gives at most 1.5 bits of information. Let us assume the value of $H_L \approx 1.25$ bits. Thus the redundancy of the English language is

$$R_{Eng} = 1 - \frac{H_L}{\log_2 26} = 1 - \frac{1.25}{4.70} \approx 0.75. \tag{1.20}$$

This suggests that, theoretically, the English text can be losslessly compressed to one fourth of its size. It is interesting that most natural languages in the world have similar redundancies.

Let us now briefly consider the redundancy in spoken English. Suppose an average speaker speaks 60 words per minute and the average number of letters per word is 6. The average number of letters spoken per second in this case is 6 letters/sec. Assuming each letter carries 1.25 bits of information, the information rate of an average speaker is 7.5 bits/sec. If each letter is represented by 5 bits, the bit rate of an average speaker is 30 bits/sec. However, the typical data rate requirement for speech is 32 kilobits/sec.

**Definition 1.6** The **Average Conditional Self-Information called the Conditional Entropy** is defined as

$$H(X|Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} P(x_i, y_j) \log \frac{1}{P(x_i|y_j)}. \tag{1.21}$$

The physical interpretation of this definition is as follows. $H(X|Y)$ is the information (or uncertainty) in $X$ after $Y$ is observed. Based on the definitions of $H(X|Y)$ and $H(X|Y)$ we can write

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{1.22}$$

We make the following observations:

(i) Since $I(X; Y) \geq 0$, it implies that $H(X) \geq H(X|Y)$.

(ii) The case $I(X; Y) = 0$ implies that $H(X) = H(X|Y)$, which is possible if and only if $X$ and $Y$ are statistically independent.

(iii) Since $H(X|Y)$ is the average amount of uncertainty (information) in $X$ after we observe $Y$ and $H(X)$ is the average amount of uncertainty (self-information) of $X$, $I(X; Y)$ is the average amount of uncertainty (mutual information) about $X$ having observed $Y$.

(iv) Since $H(X) \geq H(X|Y)$, the observation of $Y$ does not increase the entropy (uncertainty). It can only decrease the entropy. That is, observing $Y$ cannot reduce the information about $X$, it can only add to the information.

**Definition 1.7** The **Joint Entropy** of a pair of discrete random variables $(X, Y)$ with a joint distribution $p(x, y)$ is defined as

$$H(X; Y) = -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log p(x_i, y_j). \tag{1.23}$$

By using the mathematical definitions of $H(X)$, $H(X, Y)$ and $H(X|Y)$ we obtain the following chain rule

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y). \quad (1.24)$$

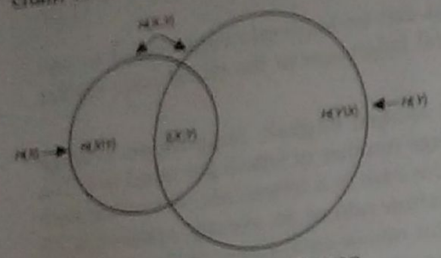

Fig. 1.5 The relationship between entropy and the mutual information.

From (1.22) and (1.24) we obtain

$$I(X; Y) = H(X) + H(Y) - H(X, Y). \quad (1.25)$$

Finally, we note that

$$I(X; X) = H(X) - H(X|X) = H(X). \quad (1.26)$$

This explains why, in Definition 1.5, the average self information is also called the entropy. The relationship between entropy and the mutual information can be expressed neatly using a Venn diagram, as depicted in Fig. 1.5.

**Example 1.8** Consider the BSC discussed in Example 1.3. Let the input symbols be '0' with probability $q$ and '1' with probability $1 - q$ as shown in Fig. 1.6.

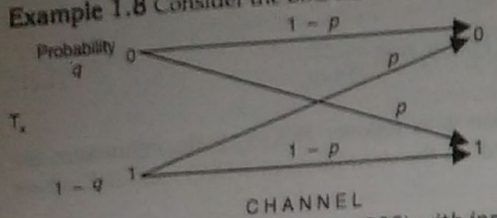


Fig. 1.6 A binary symmetric channel (BSC) with input symbols probabilities equal to $q$ and $1 - q$.

The entropy of this binary source is

$$H(X) = -\sum_{i=0}^{1} P(x_i) \log P(x_i) =$$

$$-q \log_2(q) - (1-q) \log_2(1-q).$$

The conditional entropy is given by

$$H(X|Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} P(x_i, y_j) \log \frac{1}{P(x_i | y_j)}. \quad (1.27)$$

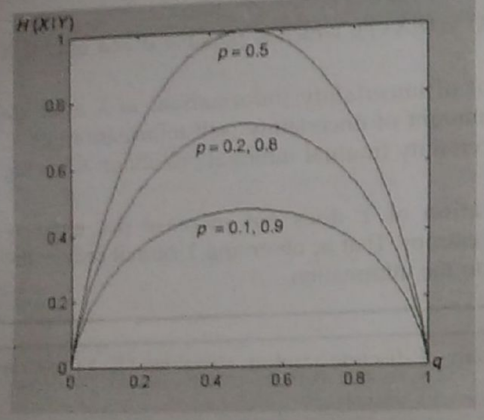


Fig. 1.7 The plot of the conditional entropy $H(X|Y)$ versus $q$.

In order to calculate the values of $H(X|Y)$, we can make use of the following equalities

$$P(x_i, y_j) = P(x_i | y_j)\, P(y_j) = P(y_j | x_i)\, P(x_i). \quad (1.28)$$

The plot of $H(X|Y)$ versus $q$ is given in Fig. 1.7 with $p$ as the parameter.

The average mutual information $I(X; Y)$ is given in Fig. 1.8. It can be seen from the plot that as we increase the parameter $p$ from 0 to 0.5, $I(X; Y)$



decreases. Physically it implies that, as we make the channel less reliable (increase the value of $p \le 0.5$), the mutual information between the random variable $X$ (at the transmitter end) and the random variable $Y$ (receiver end) decreases.

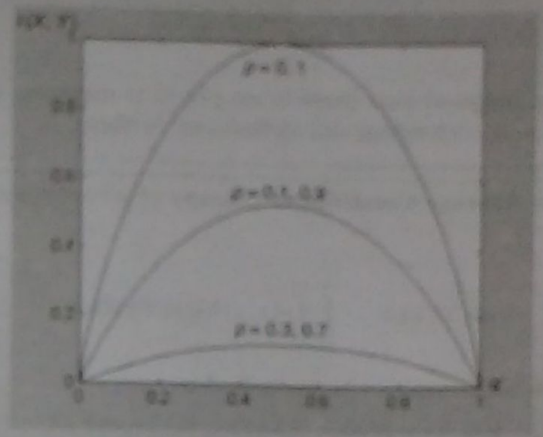

Fig. 1.8 The plot of the average mutual information $I(X; Y)$ versus $q$.

## 1.4 INFORMATION MEASURES FOR CONTINUOUS RANDOM VARIABLES

The definitions of mutual information for discrete random variables can be directly extended to continuous random variables. Let $X$ and $Y$ be random variables with joint probability density function (pdf) $p(x, y)$ and marginal pdfs $p(x)$ and $p(y)$. The average mutual information between $X$ and $Y$ is defined as follows.

**Definition 1.8** The **Average Mutual Information** between two continuous random variables $X$ and $Y$ is defined as

$$I(X; Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x) p(y|x) \log \frac{p(y|x) p(x)}{p(x) p(y)} dx dy. \quad (1.29)$$

It should be pointed out that the definition of average mutual information can be carried over from discrete random variables to the continuous random variables, but the concept and physical interpretation cannot. The reason is that the information content in a continuous random variable is actually infinite, and we require infinite number of bits to represent a continuous random variable precisely. The self-information and hence the entropy is infinite. To get around the problem we define a quantity called the differential entropy.

**Definition 1.9** The **Differential Entropy** of a continuous random variable $X$ is defined as

$$h(X) = -\int_{-\infty}^{\infty} p(x)\log p(x)\,. \tag{1.30}$$

Again, it should be understood that there is no physical meaning attached to the above quantity. We carry on with extending our definitions further.

**Definition 1.10** The **Average Conditional Entropy** of a continuous random variable $X$ given $Y$ is defined as

$$h(X|Y) = -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x,y)\log p(x|y)dxdy\,. \tag{1.31}$$

The average mutual information can be expressed as

$$I(X;Y) = h(X) - h(X|Y) = h(Y) - h(Y|X). \tag{1.32}$$

## 1.5 SOURCE CODING THEOREM

In this section we wish to explore the efficient representation (efficient coding) of symbols generated by a source. The primary motivation is the compression of data due to efficient representation of the symbols. Suppose a discrete memoryless source (DMS) outputs a symbol every $t$ seconds. Each symbol is selected from a finite set of symbols $x_i$, $i = 1, 2, ..., L$, occurring with probabilities $P(x_i)$, $i = 1, 2, ..., L$. The entropy of this DMS in bits per source symbols is

$$H(X) = -\sum_{j=1}^{L} P(x_i)\log_2 P(x_i) \le \log_2 L\,. \tag{1.33}$$

The equality holds when the symbols are equally likely. It implies that the average number of bits per source symbol is $H(X)$ and the source rate is $H(X)/t$ bits/sec.

Now let us suppose that we wish to represent the 26 letters in the English alphabet using bits. We observe that $2^5 = 32 > 26$. Hence, each of the letters can be uniquely represented using 5 bits. This is an example of a **Fixed Length Code (FLC)**. Each letter has a corresponding 5 bit long **codeword**.

**Definition 1.11** A code is a set of vectors called **codewords**.

Suppose a discrete memoryless source (DMS) outputs a symbol selected from a finite set of symbols $x_i$, $i = 1, 2, ..., L$. The number of binary digits (bits) $R$ required for unique coding, when $L$ is a power of 2 is



$$R = \log_2 L, \tag{1.34}$$

and, when $L$ is not a power of 2, it is

$$R = \lfloor \log_2 L \rfloor + 1. \tag{1.35}$$

As we saw earlier, to encode the letters of the English alphabet, we need $R = \lfloor \log_2 26 \rfloor + 1 = 5$ bits. The fixed length code for the English alphabet suggests that each of the letters in the alphabet is equally important (probable) and hence each one requires 5 bits for representation. However, we know that some of the letters are less common (x, q, z etc.) while some others are more frequently used (s, t, e etc.). It appears that allotting equal number of bits to both the frequently used letters as well as not so commonly used letters is *not* an efficient way of representation (coding). Intuitively, we should represent the more frequently occurring letters by fewer number of bits and represent the less frequently occurring letters by larger number of bits. In this manner, if we have to encode a whole page of written text, we might end up using fewer number of bits overall. When the source symbols are not equally probable, a more efficient method to use is a **Variable Length Code (VLC)**.

**Example 1.9** Suppose we have only the first eight letters of the English alphabet (A – H) in our vocabulary. The fixed length code for this set of letters would be

Fixed length code

| *Letter* | *Codeword* | *Letter* | *Codeword* |
|---|---|---|---|
| A | 000 | E | 100 |
| B | 001 | F | 101 |
| C | 010 | G | 110 |
| D | 011 | H | 111 |

A variable length code for the same set of letters can be

Variable length code 1

| *Letter* | *Codeword* | *Letter* | *Codeword* |
|---|---|---|---|
| A | 00 | E | 101 |
| B | 010 | F | 110 |
| C | 011 | G | 1110 |
| D | 100 | H | 1111 |

Suppose we have to code the series of letters: "A BAD CAB". The fixed length and the variable length representation of the pseudo-sentence would be

| | | |
|---|---|---|
| Fixed Length Code | 000 001 000 011 010 000 001 | Total bits = 21 |
| Variable Length Code | 00 010 00 100 011 00 010 | Total bits = 18 |

Note that the variable length code uses fewer numbers of bits simply because the letters appearing more frequently in the pseudo-sentence are represented with fewer numbers of bits.

We look at yet another variable length code for the first 8 letters of the English alphabet:

Variable length code 2

| Letter | Codeword | Letter | Codeword |
|---|---|---|---|
| A | 0 | E | 10 |
| B | 1 | F | 11 |
| C | 00 | G | 000 |
| D | 01 | H | 111 |

This second variable length code appears to be more efficient in terms of representation of the letters.

| | | |
|---|---|---|
| Variable Length Code 1 | 00 010 00 100 011 00 010 | Total bits = 18 |
| Variable Length Code 2 | 0 1001 0001 | Total bits = 9 |

However there is a problem with VLC2. Consider the sequence of bits 0 1001 0001 which is used to represent A BAD CAB in the second variable length coding scheme. We could regroup the bits in a different manner to have [0] [10][0][1] [0][0][01] which translates to A EAB AAD or we can decode the vector as [0] [1][0][0][1] [0][0][0][1] which stands for A BAAB AAAB ! Obviously there is a problem with the unique decoding of the code. We have no clue where the codeword of one letter (symbol) ends and where the next one begins, since the lengths of the codewords are variable. However, this problem does not exist with the VLC1. It can be seen that no codeword forms the prefix of any other codeword. This is called the **prefix condition**. So, as soon as a sequence of bits corresponding to any one of the possible codewords is detected, we can declare that symbol decoded. Such codes are called **Instantaneous Codes**. There is no decoding delay in these codes. If decoding delays are permitted, we may use **Uniquely Decodable Codes** in which the encoded string could be generated by only one possible input string. However, one may have to wait until the entire string is obtained before decoding even the first symbol. In this example, the VLC2 is *not* a uniquely decodable code, hence not a code of any practical utility. The VLC1 is uniquely decodable, though less economical in terms of bits per symbol.

**Definition 1.12** A Prefix Code is one in which no codeword forms the prefix of any other codeword. Such codes are also called **Instantaneous Codes**.

Our objective now is to devise a systematic procedure for constructing uniquely decodable, variable length codes that are efficient in terms of average number of bits per source letter. Let the source output a symbol from a finite set of symbols $x_i$, $i = 1, 2, ..., L$, occurring with probabilities $P(x_i)$, $i = 1, 2, ..., L$. The average number of bits per source letter is defined as



$$\bar{R} = \sum_{i=1}^{L} n_i P(x_i). \tag{1.36}$$

where $n_i$ is the length of the codeword (in terms of number of bits) for the symbol $x_i$.

**Theorem 1.1 (Kraft Inequality)** A necessary and sufficient condition for the existence of a binary code with codewords having lengths $n_1 \le n_2 \le ... \, n_L$ that satisfy the prefix condition is

$$\sum_{k=1}^{L} 2^{-n_k} \le 1. \tag{1.37}$$

***Proof*** First we prove the sufficient condition. Consider a binary of order tree of order (depth) $n = n_L$. This tree has $2^n$ terminal nodes as depicted in Fig. 1.9. Let us select any code of order $n_1$ as the first codeword $c_1$. Since no codeword is the prefix of any other codeword (the prefix condition), this choice eliminates $2^{n-n_1}$ terminal codes. This process continues until the last codeword is assigned at the terminal node $n = n_L$. Consider the node of order $j < L$. The fraction of number of terminal nodes eliminated is

$$\sum_{k=1}^{j} 2^{-n_k} < \sum_{k=1}^{L} 2^{-n_k} \le 1. \tag{1.38}$$

Thus, we have been able to construct a prefix code that is embedded in the full tree of $n_L$ nodes. The nodes that are eliminated are depicted by the dotted arrow lines leading on to them in Fig. 1.9.

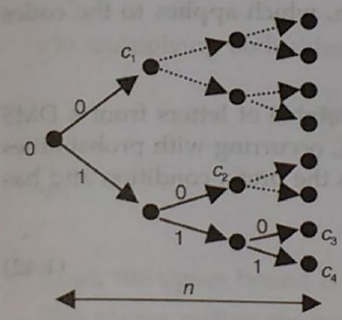


**Fig. 1.9** A binary tree of order $n_L$.

We now prove the necessary condition. We observe that in the code tree of the order $n = n_L$, the number of terminal nodes eliminated from the total number of $2^n$ terminal nodes is

$$\sum_{k=1}^{L} 2^{n-n_k} \le 2^n. \tag{1.39}$$

This leads to

$$\sum_{k=1}^{L} 2^{-n_k} \le 1. \tag{1.40}$$

We can easily extend this proof for prefix codes over an alphabet of size $M$. For the proof we will have to consider an $M$-ary tree instead of a binary tree. The inequality in this case would become

$$\sum_{k=1}^{L} M^{-n_k} \le 1. \tag{1.41}$$

**Example 1.10** *Consider the construction of a prefix code using a binary tree.*

*We start from the mother node and proceed toward the terminal nodes of the binary tree (Fig. 1.10). Let the mother node be labeled '0' (could have been labeled '1' as well). There are always two branches emanating from each node (binary tree). Let us label the upper branch '0' and the lower branch '1' (these labels could have also been mutually exchanged). First we follow the upper branch from the mother node. We obtain our first codeword $c_1 = 0$ terminating at node $n_{00}$. Since we want to construct a prefix code where no codeword is a prefix of any other codeword, we must discard all the daughter nodes generated as a result of the node labeled $c_1$. We now proceed on the lower branch from the mother node and reach the node $n_{01}$. We proceed along the upper branch first and reach node $n_{010}$. We label this as the codeword $c_2 = 10$ (the labels of the branches that lead up to this node travelling from the mother node). Following the lower branch from the node $n_{01}$, we ultimately reach the terminal nodes $n_{0110}$ and $n_{0111}$, which correspond to the codewords $c_3 = 110$ and $c_4 = 111$ respectively. Thus the binary tree has given us four prefix codewords: {0, 10, 110, 111}. By construction, this is a prefix code. For this code*

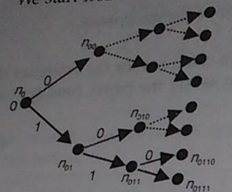


*Fig. 1.10 Constructing a binary prefix code using a binary tree.*

$$\sum_{k=1}^{L} 2^{-n_k} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 0.5 + 0.25 + 0.125 + 0.125 = 1.$$

*Hence the Kraft inequality is satisfied.*

*We now state and prove the noiseless Source Coding theorem, which applies to the codes that satisfy the prefix condition.*

**Theorem 1.2 (Source Coding Theorem)** *Let $X$ be the ensemble of letters from a DMS with finite entropy $H(X)$ and the output symbols $x_k$, $k = 1, 2, ..., L$, occurring with probabilities $P(x_k)$, $k = 1, 2, ..., L$. It is possible to construct a code that satisfies the prefix condition and has an average length $\overline{R}$ that satisfies the inequality*

$$H(X) \le \overline{R} < H(X) + 1. \tag{1.42}$$

***Proof*** *First consider the lower bound of the inequality. For codewords that have length $n_k$, $1 \le k \le L$, the difference $H(X) - \overline{R}$ can be expressed as*

$$H(X) - \overline{R} = \sum_{k=1}^{L} P(x_k) \log_2 \frac{1}{P(x_k)} - \sum_{k=1}^{L} P(x_k) n_k = \sum_{k=1}^{L} P(x_k) \log_2 \frac{2^{-n_k}}{P(x_k)}.$$



We now make use of the inequality $\ln x \le x - 1$ to get

$$H(x) - \overline{R} \le (\log_2 e) \sum_{k=1}^{L} P(x_k) \left( \frac{2^{-n_k}}{p(x_k)} - 1 \right)$$

$$\le (\log_2 e) \left( \sum_{k=1}^{L} 2^{-n_k} - 1 \right) \le 0.$$

The last inequality follows from the Kraft inequality. Equality holds, if and only if, $P(x_k) = 2^{-n_k}$ for $1 \le k \le L$. Thus the lower bound is proved.

We now prove the upper bound. Let us select the codeword lengths $n_k$ such that

First consider $$2^{-n_k} \le P(x_k) < 2^{-n_k} + 1.$$

Summing both sides over $1 \le k \le L$ gives us $$2^{-n_k} \le P(x_k).$$

$$\sum_{k=1}^{L} 2^{-n_k} \le \sum_{k=1}^{L} P(x_k) = 1,$$

which is the Kraft inequality for which there exist a code satisfying the prefix condition.

Next consider $$P(x_k) < 2^{-n_k+1}.$$

Take logarithm on both sides to get

or, $$\log_2 P(x_k) < -n_k + 1$$

$$n_k < 1 - \log_2 P(x_k).$$

On multiplying both sides by $P(x_k)$ and summing over $1 \le k \le L$ we obtain

$$\sum_{k=1}^{L} P(x_k) n_k < \sum_{k=1}^{L} P(x_k) + \left( -\sum_{k=1}^{L} P(x_k) \log_2 P(x_k) \right),$$

or,

$$\overline{R} < H(X) + 1.$$

Thus, the upper bound is proved.

The source coding theorem tells us that for any prefix code used to represent the symbols from a source, the *minimum* number of bits required to represent the source symbols *on an average* must be at least equal to the entropy of the source. If we have found a prefix code that satisfies $\overline{R} = H(X)$ for a certain source $X$, we must abandon further search because we cannot do any better. The theorem also tells us that a source with a higher entropy (uncertainty) requires, on an average, more number of bits to represent the source symbols in terms of a prefix code.

**Definition 1.13** The Efficiency of a prefix code is defined as

$$\eta = \frac{H(X)}{\overline{R}} \tag{1.43}$$

It is clear from the source coding theorem that the efficiency of a prefix code is $\eta \leq 1$. Efficient representation of symbols leads to **compression** of data. Source coding is primarily used for compression of data (speech, text, image, video etc.).

**Example 1.11** Consider a source $X$ which generates four symbols with probabilities $P(x_1) = 0.5$, $P(x_2) = 0.3$, $P(x_3) = 0.1$ and $P(x_4) = 0.1$. The entropy of this source is

$$H(X) = -\sum_{k=1}^{4} P(x_k)\log_2 P(x_k) = 1.685 \text{ bits.}$$

Suppose we use the prefix code {0, 10, 110, 111} constructed in Example 1.10. Then the average codeword length, $\overline{R}$ is given by

$$\overline{R} = \sum_{k=1}^{4} n_k P(x_k) = 1(0.5) + 2(0.3) + 3(0.1) + 3(0.1) = 1.700 \text{ bits.}$$

Thus we have

$$H(X) \leq \overline{R} \leq H(X) + 1.$$

The efficiency of this code is $\eta = (1.685/1.700) = 0.9912$. Had the source symbol probabilities been $P(x_k) = 2^{-n_k}$ i.e., $P(x_1) = 2^{-1} = 0.5$, $P(x_2) = 2^{-2} = 0.25$, $P(x_3) = 2^{-3} = 0.125$ and $P(x_4) = 2^{-3} = 0.125$, 0.125, the average codeword length, $= 1.750$ bits $= H(X)$. In this case, $\eta = 1$.

## 1.6 HUFFMAN CODING

We will now study an algorithm for constructing efficient source codes for a DMS with source symbols that are not equally probable. A variable length encoding algorithm was suggested by Huffman in 1952, based on the source symbol probabilities $P(x_i)$, $i = 1, 2, \ldots, L$. The algorithm is optimal in the sense that the average number of bits require to represent the source symbols is a minimum provided the prefix condition is met. The steps of the Huffman coding algorithm are given below.

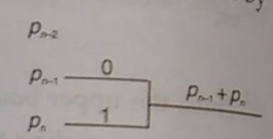


**Fig. 1.11** Combining probabilities in Huffman coding.

(i) Arrange the source symbols in a *decreasing* order of their probabilities.

(ii) Take the bottom two symbols and tie them together as shown in Fig. 1.11. Add the probabilities of the two symbols and write it on the combined node. Label the two branches with a '1' and a '0' as depicted in Fig. 1.11.





(iii) Treat this sum of probabilities as a new probability associated with a new symbol. Again pick the two smallest probabilities, tie them together to form a new probability. Each time we perform the combination of two symbols we reduce the total number of symbols by one. Whenever we tie together two probabilities (nodes) we label the two branches with a '1' and a '0'.

(iv) Continue the procedure until only one probability is left (and it should be 1 if your addition is correct). This completes the construction of the Huffman tree.

(v) To find out the prefix codeword for any symbol, follow the branches from the final node back to the symbol. While tracing back the route read out the labels on the branches. This is the codeword for the symbol.

The algorithm can be easily understood using the following example.

**Example 1.12** Consider a DMS with seven possible symbols $x_i$, $i = 1, 2, \ldots, 7$ and the corresponding probabilities $P(x_1) = 0.37$, $P(x_2) = 0.33$, $P(x_3) = 0.16$, $P(x_4) = 0.07$, $P(x_5) = 0.04$, $P(x_6) = 0.02$, and $P(x_7) = 0.01$. We first arrange the probabilities in decreasing order and then construct the Huffman tree.

| *Symbol* | *Probability* | *Self-Information* | *Codeword* |
|---|---|---|---|
| $x_1$ | 0.37 | 1.4344 | 0 |
| $x_2$ | 0.33 | 1.5995 | 10 |
| $x_3$ | 0.16 | 2.6439 | 110 |
| $x_4$ | 0.07 | 3.8365 | 1110 |
| $x_5$ | 0.04 | 4.6439 | 11110 |
| $x_6$ | 0.02 | 5.6439 | 111110 |
| $x_7$ | 0.01 | 6.6439 | 111111 |

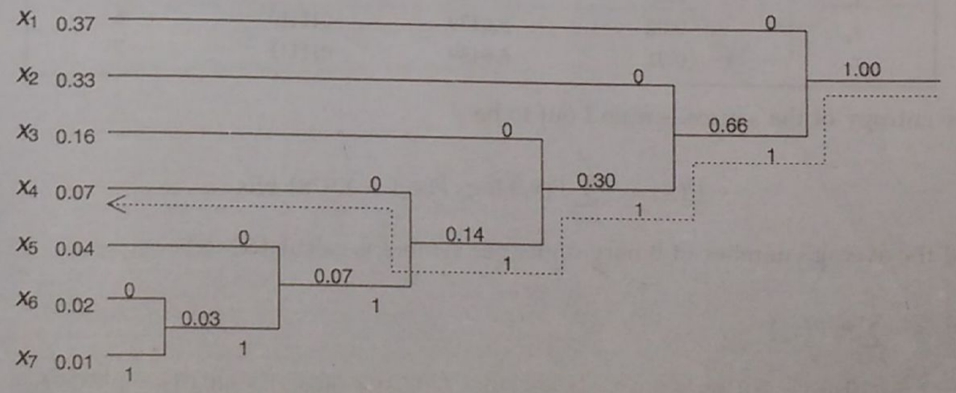


**Fig. 1.12** Huffman coding for Example 1.12.

To find the codeword for any particular symbol, we just trace back the route from the final node to the symbol. For the sake of illustration we show the route for the symbol $x_4$ with probability 0.07 with the dotted line. We read out the labels of the branches on the way to obtain the codeword as 1110.

The entropy of the source is found out to be

$$H(X) = -\sum_{k=1}^{7} P(x_k)\log_2 P(x_k) = 2.1152 \text{ bits},$$

and the average number of binary digits per symbol is calculated to be

$$\overline{R} = \sum_{k=1}^{7} n_k P(x_k)$$
$$= 1(0.37) + 2(0.33) + 3(0.16) + 4(0.07) + 5(0.04) + 6(0.02) + 6(0.01)$$
$$= 2.1700 \text{ bits}.$$

The efficiency of this code is $\eta = (2.1152/2.1700) = 0.9747$.

**Example 1.13** This example shows that Huffman coding is not unique. Consider a DMS with seven possible symbols $x_i$, $i = 1, 2, \ldots, 7$ and the corresponding probabilities $P(x_1) = 0.46$, $P(x_2) = 0.30$, $P(x_3) = 0.12$, $P(x_4) = 0.06$, $P(x_5) = 0.03$, $P(x_6) = 0.02$, and $P(x_7) = 0.01$.

| *Symbol* | *Probability* | *Self Information* | *Codeword* | *Codeword Length* |
|---|---|---|---|---|
| $x_1$ | 0.46 | 1.1203 | 1 | 1 |
| $x_2$ | 0.30 | 1.7370 | 00 | 2 |
| $x_3$ | 0.12 | 3.0589 | 010 | 3 |
| $x_4$ | 0.06 | 4.0589 | 0110 | 4 |
| $x_5$ | 0.03 | 5.0589 | 01110 | 5 |
| $x_6$ | 0.02 | 5.6439 | 011110 | 6 |
| $x_7$ | 0.01 | 6.6439 | 011111 | 7 |

The entropy of the source is found out to be

$$H(X) = -\sum_{k=1}^{7} P(x_k)\log_2 P(x_k) = 1.9781 \text{ bits}$$

and the average number of binary digits per symbol is calculated to be

$$\overline{R} = \sum_{k=1}^{7} n_k P(x_k)$$
$$= 1(0.46) + 2(0.30) + 3(0.12) + 4(0.06) + 5(0.03) + 6(0.02) + 6(0.01) = 1.9900 \text{ bits}$$

The efficiency of this code is $\eta = (1.9781/1.9900) = 0.9940$.

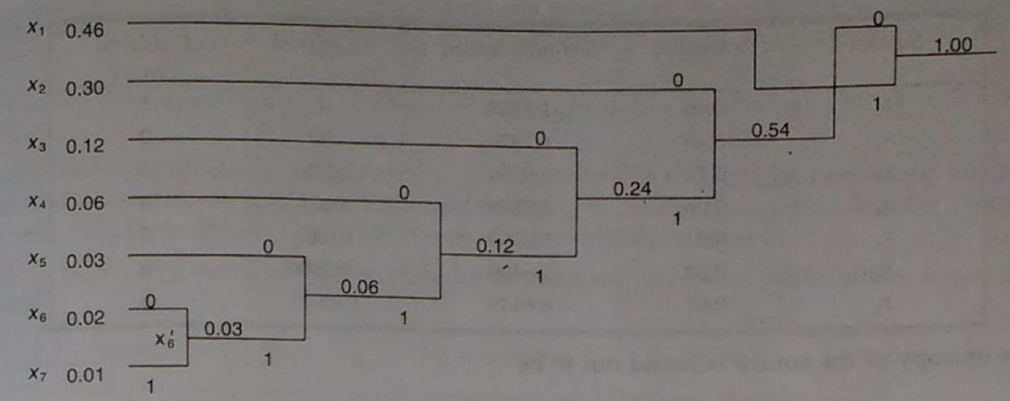


**Fig. 1.13** Huffman coding for Example 1.13.

We shall now see that Huffman coding is not unique. Consider the combination of the two smallest probabilities (symbols $x_6$ and $x_7$). Their sum is equal to 0.03, which is equal to the next higher probability corresponding to the symbol $x_5$. So, for the second step, we may choose to put this combined probability (belonging to say, symbol $x_6'$) higher or lower than, the symbol $x_5$. Suppose we put the combined probability at a lower level. We proceed further, to again find the combination of $x_6'$ and $x_5$ yield the probability 0.06, which is equal to that of symbol $x_4$. We again have a choice whether to put the combined probability higher or lower than, the symbol $x_4$. Each time we make a choice (or flip a fair coin) we end up changing the final codeword for the symbols. In Fig. 1.14, each time we have to make a choice between two probabilities that are equal, we put the probability of the combined symbols at a higher level.

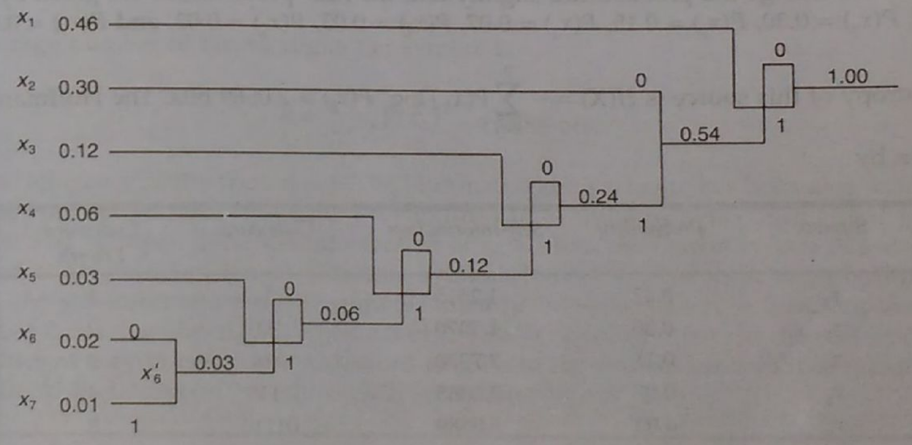


**Fig. 1.14** Alternate way of Huffman coding in Example 1.13 which leads to a different code.

| Symbol | Probability | Self-information | Codeword | Codeword Length |
|---|---|---|---|---|
| $x_1$ | 0.46 | 1.1203 | 1 | 1 |
| $x_2$ | 0.30 | 1.7370 | 00 | 2 |
| $x_3$ | 0.12 | 3.0589 | 011 | 3 |
| $x_4$ | 0.06 | 4.0589 | 0101 | 4 |
| $x_5$ | 0.03 | 5.0589 | 01001 | 5 |
| $x_6$ | 0.02 | 5.6439 | 010000 | 6 |
| $x_7$ | 0.01 | 6.6439 | 010001 | 6 |

The entropy of the source is found out to be

$$H(X) = -\sum_{k=1}^{7} P(x_k) \log_2 P(x_k) = 1.9781 \text{ bits},$$

and the average number of binary digits per symbol is calculated to be

$$\overline{R} = \sum_{k=1}^{7} n_k P(x_k)$$
$$= 1(0.46) + 2(0.30) + 3(0.12) + 4(0.06) + 5(0.03) + 6(0.02) + 6(0.01)$$
$$= 1.9900 \text{ bits}.$$

The efficiency of this code is $\eta = (1.9781/1.9900) = 0.9940$. Thus, both the codes are equally efficient. This tells us that Huffman codes are not unique for a given set of probabilities.

Suppose we change the probabilities slightly and the new probabilities are given by $P(x_1) = 0.42$, $P(x_2) = 0.30$, $P(x_3) = 0.15$, $P(x_4) = 0.07$, $P(x_5) = 0.03$, $P(x_6) = 0.02$, and $P(x_7) = 0.01$.

The entropy of this source is $H(X) = -\sum_{k=1}^{7} P(x_k) \log_2 P(x_k) = 2.0569$ bits. The Huffman code is given by

| Symbol | Probability | Self-Information | Codeword | Codeword Length |
|---|---|---|---|---|
| $x_1$ | 0.42 | 1.2515 | 1 | 1 |
| $x_2$ | 0.30 | 1.7370 | 00 | 2 |
| $x_3$ | 0.15 | 2.7370 | 010 | 3 |
| $x_4$ | 0.07 | 3.8365 | 0110 | 4 |
| $x_5$ | 0.03 | 5.0589 | 01110 | 5 |
| $x_6$ | 0.02 | 5.6439 | 011110 | 6 |
| $x_7$ | 0.01 | 6.6439 | 011111 | 6 |



We observe that in spite of the change in the probability values, the Huffman code is the same as generated by Fig. 1.13. The average codeword length $\overline{R} = \sum_{n=1}^{7} n_k P(x_k) =$ 2.0800 bits. The efficiency of this code is $\eta = (2.0569/2.0800) = 0.9889$. Intuitively, we can say that the codeword lengths, which *must* be integers, cannot change to reflect the change in probabilities (and hence the self-information) of the input signals.

Let us now carry out a similar exercise for another set of symbol probabilities as shown below.

| Symbol | Probability | Self-Information | Codeword | Codeword Length |
|---|---|---|---|---|
| $x_1$ | $1/2$ | 1 | 1 | 1 |
| $x_2$ | $1/2^2$ | 2 | 00 | 2 |
| $x_3$ | $1/2^3$ | 3 | 010 | 3 |
| $x_4$ | $1/2^4$ | 4 | 0110 | 4 |
| $x_5$ | $1/2^5$ | 5 | 01110 | 5 |
| $x_6$ | $1/2^6$ | 6 | 011110 | 6 |
| $x_7$ | $1/2^6$ | 6 | 011111 | 6 |

A probability distribution is called **D–adic** with respect to $D$, if each of the probabilities is equal to $D^{-n}$ for some integer $n$. By definition, the distribution in the above table is $D$–adic. The entropy of this source is

$$H(X) = -\sum_{k=1}^{7} P(x_k) \log_2 P(x_k) = 1.9688 \text{ bits}.$$

The average number of binary digits per symbol is

$$\overline{R} = \sum_{k=1}^{7} n_k P(x_k) = 1.9688 \text{ bits}.$$

Thus the efficiency of the code $\eta = 1$. The Huffman coding scheme has been able to match the codeword lengths *exactly* to the probabilities (and thus the self-information) of the symbols. For example, if the self-information of $x_5$ is 5 bits, the Huffman code has actually allocated a codeword of length 5 to this symbol. It is clear that to achieve optimality ($\eta = 1$), the self-information of the symbols must be integers, which in turn, implies that the probabilities must be negative powers of 2. This is not always true in the real world. A more efficient way to match the codeword lengths to the symbol probabilities is done by using the Arithmetic code, which we will study later in this chapter.

---

In the above examples, encoding is done symbol by symbol. A more efficient procedure is to encode blocks of $B$ symbols at a time. In this case, the bounds of the source coding theorem become

$$BH(X) \le \overline{R}_B < BH(X)+1, \tag{1.44}$$

since the entropy of a $B$–symbol block is simply $BH(X)$, and $\overline{R}_B$ is the average number of bits per $B$–symbol block. We can rewrite the bound as

$$H(X) \le \frac{\overline{R}_B}{B} < H(X)+\frac{1}{B}, \tag{1.45}$$

where $\frac{\overline{R}_B}{B} \equiv \overline{R}$ is the average number of bits per source symbol. Thus, $\overline{R}$ can be made arbitrarily close to $H(X)$ by selecting a large enough block $B$.

**Example 1.14** Consider the source symbols and their respective probabilities listed below.

| *Symbol* | *Probability* | *Self-Information* | *Codeword* |
|---|---|---|---|
| $x_1$ | 0.40 | 1.3219 | 1 |
| $x_2$ | 0.35 | 1.5146 | 00 |
| $x_3$ | 0.25 | 2.0000 | 01 |

For this code, the entropy of the source is

$$H(X) = -\sum_{k=1}^{3} P(x_k)\log_2 P(x_k) = 1.5589 \text{ bits.}$$

The average number of binary digits per symbol is

$$\overline{R} = \sum_{k=1}^{3} n_k P(x_k)$$

$$= 1(0.40) + 2(0.35) + 2(0.25) = 1.60 \text{ bits.}$$

and the efficiency of this code is $\eta = (1.5589/1.6000) = 0.9743$.

We now group together the symbols, two at a time, and again apply the Huffman encoding algorithm. The probabilities of the symbol pairs, in decreasing order, are listed below.

| *Symbol* | *Probability* | *Self-information* | *Codeword* |
|---|---|---|---|
| $x_1x_1$ | 0.1600 | 2.6439 | 10 |
| $x_1x_2$ | 0.1400 | 2.8365 | 001 |
| $x_2x_1$ | 0.1400 | 2.8365 | 010 |
| $x_2x_2$ | 0.1225 | 3.0291 | 011 |
| $x_1x_3$ | 0.1000 | 3.3219 | 111 |



| | | | |
|---|---|---|---|
| $x_3x_1$ | 0.1000 | 3.3219 | 0000 |
| $x_2x_3$ | 0.0875 | 3.5146 | 0001 |
| $x_3x_2$ | 0.0875 | 3.5146 | 1100 |
| $x_3x_3$ | 0.0625 | 4.0000 | 1101 |

For this code, the entropy is

$$2H(X) = -\sum_{k=1}^{9} P(x_k)\ \log_2 P(x_k) = 3.1177 \text{ bits}$$

$$\Rightarrow H(X) = 1.5589 \text{ bits.}$$

Note that the source entropy has not changed. The average number of binary digits per block (symbol pair) is

$$\overline{R}_B = \sum_{k=1}^{9} n_k P(x_k)$$

$$= 2(0.1600) + 3(0.1400) + 3(0.1400) + 3(0.1225) + 3(0.1000) + 4(0.1000) + 4(0.0875) + 4(0.0875) + 4(0.0625) = 3.1775 \text{ bits per symbol pair}$$

$$\Rightarrow \overline{R} = 3.1775/2 = 1.5888 \text{ bits per symbol.}$$

and the efficiency of this code is $\eta = (1.5589/1.5888) = 0.9812$. Thus we see that grouping of two letters to make a symbol has improved the coding efficiency.

**Example 1.15** Consider the source symbols and their respective probabilities listed below.

| *Symbol* | *Probability* | *Self-Information* | *Codeword* |
|---|---|---|---|
| $x_1$ | 0.50 | 1.0000 | 1 |
| $x_2$ | 0.30 | 1.7370 | 00 |
| $x_3$ | 0.20 | 2.3219 | 01 |

For this code, the entropy of the source is

$$H(X) = -\sum_{k=1}^{3} P(x_k)\log_2 P(x_k) = 1.4855 \text{ bits.}$$

The average number of binary digits per symbol is

$$\overline{R} = \sum_{k=1}^{3} n_k P(x_k)$$

$$= 1(0.50) + 2(0.30) + 2(0.20) = 1.50 \text{ bits}$$

and the efficiency of this code is $\eta = (1.4855\ /1.5000) = 0.9903$.

We now group together the symbols, two at a time, and again apply the Huffman encoding algorithm. The probabilities of the symbol pairs, in decreasing order, are listed below.

| Symbol | Probability | Self-Information | Codeword |
|---|---|---|---|
| $x_1x_1$ | 0.25 | 2.0000 | 00 |
| $x_1x_2$ | 0.15 | 2.7370 | 010 |
| $x_2x_1$ | 0.15 | 2.7370 | 011 |
| $x_1x_3$ | 0.10 | 3.3219 | 100 |
| $x_3x_1$ | 0.10 | 3.3219 | 110 |
| $x_2x_2$ | 0.09 | 3.4739 | 1010 |
| $x_2x_3$ | 0.06 | 4.0589 | 1011 |
| $x_3x_2$ | 0.06 | 4.0589 | 1110 |
| $x_3x_3$ | 0.04 | 4.6439 | 1111 |

For this code, the entropy is

$$2H(X) = -\sum_{k=1}^{9} P(x_k)\log_2 P(x_k) = 2.9710 \text{ bits}$$

$$\Rightarrow H(X) = 1.4855 \text{ bits.}$$

The average number of binary digits per block (symbol pair) is

$$\bar{R}_B = \sum_{k=1}^{9} n_k P(x_k)$$

$$= 2(0.25) + 3(0.15) + 3(0.15) + 3(0.10) + 3(0.10) + 4(0.09) + 4(0.06) + 4(0.06) + 4(0.04) = 3.00 \text{ bits per symbol pair}$$

$$\Rightarrow \bar{R} = 3.00/2 = 1.5000 \text{ bits per symbol.}$$

and the efficiency of this code is $\eta_2 = (1.4855/1.5000) = 0.9903$. In this case, grouping together two letters at a time has not increased the efficiency of the code. However, if we group 3 letters at a time (triplets) and then apply Huffman coding, we obtain the code efficiency as $\eta_3 = 0.9932$. Upon grouping four letters at a time we see a further improvement ($\eta_4 = 0.9946$).

## 1.7 SHANNON–FANO–ELIAS CODING

Codes that use the codeword lengths of $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil$ are called **Shannon Codes**. Shannon codeword lengths satisfy the Kraft inequality and can therefore be used to construct a uniquely



decodable code. In this section we will study another simple method for constructing uniquely decodable codes based on the Shannon-Fano-Elias encoding technique. It uses the **Cumulative Distribution Function** to allocate the codewords. The cumulative distribution function is defined as

$$F(x) = \sum_{z \le x} P(z), \tag{1.46}$$

where $P(z)$ are the probability of occurrence of the $z$. The cumulative distribution function consists of steps of size $P(x)$, as shown in Fig. 1.15. Let us define a modified cumulative distribution function as

$$\bar{F}(x) = \sum_{z<x} P(z) + \frac{1}{2}P(x), \tag{1.47}$$

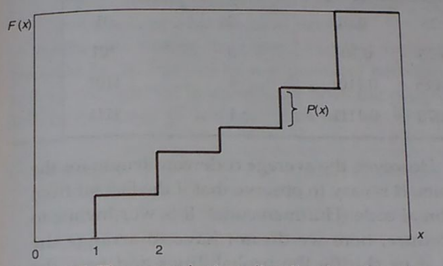


Fig. 1.15 The cumulative distribution function.

where $\bar{F}(x)$ represents the sum of the probabilities of all symbols less than $x$ plus half the probability of the symbols $x$. The value of the function $\bar{F}(x)$ is the midpoint of the step corresponding to $x$ of the cumulative distribution function. Since probabilities are positive, $F(x) \neq F(z)$ if $x \neq z$. Thus it is possible to determine $x$ given $\bar{F}(x)$ merely by looking at the graph of the cumulative distribution function. Therefore, the value of $\bar{F}(x)$ can be used to code $x$.

In general, $\bar{F}(x)$ is a real number. This means we require an infinite number of bits to represent $\bar{F}(x)$ which would lead to an inefficient code. Suppose we round off $\bar{F}(x)$ and use only the first $l(x)$ bits, denoted by $\lfloor \bar{F}(x) \rfloor_{l(x)}$. By the definition of rounding off we have

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < \frac{1}{2^{l(x)}}. \tag{1.48}$$

If $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$, then,

$$\frac{1}{2^{l(x)}} < \frac{P(x)}{2} = \bar{F}(x) - F(x-1). \tag{1.49}$$

This implies that $\lfloor \bar{F}(x) \rfloor_{l(x)}$ lies within the step corresponding to $x$, and $l(x)$ bits are sufficient to describe $x$. The interval corresponding to any codeword has length $2^{-l(x)}$. From (1.48) we see

that this interval is less than half the height of the step corresponding to $x$. Since we use $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ bits to represent $x$, the expected length of this code is

$$\bar{R} = \sum_x P(x) l(x) = \sum_x P(x) \left( \left\lceil \log \frac{1}{P(x)} \right\rceil + 1 \right) < H(X) + 2 . \quad (1.50)$$

Thus the Shannon–Fano–Elias coding scheme achieves codeword length within two bits of the entropy.

**Example 1.16** Consider the $D$–adic distribution given in the following table.

| *Symbol* | *Probability* | $F(x)$ | $\bar{F}(x)$ | $\bar{F}(x)$ *(binary)* | $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ | *Codeword* |
|---|---|---|---|---|---|---|
| $x_1$ | 1/2 | 0.5 | 0.25 | 0.01 | 2 | 01 |
| $x_2$ | $1/2^2$ | 0.75 | 0.625 | 0.101 | 3 | 101 |
| $x_3$ | $1/2^3$ | 0.875 | 0.8125 | 0.1101 | 4 | 1101 |
| $x_4$ | $1/2^3$ | 1 | 0.9375 | 0.1111 | 4 | 1111 |

The entropy for this distribution is 1.75 bits. However, the average codeword length for the Shannon-Fano-Elias coding scheme is 2.75 bits. It is easy to observe that if the last bit from all the codewords is deleted, we get the optimal code (Huffman code). It is worthwhile to note that unlike in Huffman coding procedure, here we do not have to arrange the probabilities in a descending order first. Let us shuffle the probabilities and redo the exercise.

| *Symbol* | *Probability* | $F(x)$ | $\bar{F}(x)$ | $\bar{F}(x)$ *(binary)* | $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ | *Codeword* |
|---|---|---|---|---|---|---|
| $x_1$ | $1/2^2$ | 0.25 | 0.125 | 0.001 | 3 | 001 |
| $x_2$ | 1/2 | 0.75 | 0.625 | 0.10 | 2 | 10 |
| $x_3$ | $1/2^3$ | 0.875 | 0.8125 | 0.1101 | 4 | 1101 |
| $x_4$ | $1/2^3$ | 1 | 0.9375 | 0.1111 | 4 | 1111 |

We observe that the codewords obtained from the Shannon-Fano-Elias coding procedure is not unique. The average codeword length is again 2.75 bits. However, this time we cannot get the optimal code simply by deleting the last bit from every codeword. If we do so, the code no longer remains a prefix code. The basic concept of Shannon-Fano-Elias coding is used in a computationally efficient algorithm for encoding and decoding called Arithmetic Coding.



## 1.8 ARITHMETIC CODING

As we have seen from Example 1.13, Huffman codes are only optimal if the probabilities of the symbols are negative powers of two. This is because all prefix codes work at the bit level. There are two ways to look at it.

(a) Prefix codes-try to match the self-information of the symbols using codewords whose lengths are integers. The length matching may ascribe a codeword either a longer than the self-information or shorter. The exact match is possible, if and only if, the self-information is in integral number of bits.

(b) If we consider the prefix codes being generated using a binary tree, the decisions between tree branches always take one bit. This is regardless of the fact whether the probabilities for the branches are 0.5/0.5 or 0.9/0.1. In the latter case it would theoretically take only 0.15 bits ($-\log_2(0.9)$) to select the first branch and 3.32 bits ($-\log_2(0.1)$) to select the second branch, making the average code length 0.467 bits ($0.9 \times 0.15 + 0.1 \times 3.32$). The Huffman code still needs one bit for each decision.

Arithmetic coding does not have this restriction. It works by representing the file to be encoded by an interval of real numbers between 0 and 1. Successive symbols in the message reduce this interval in accordance with the probability of that symbol. The more likely symbols reduce the range by less, and thus add fewer bits to the message.

**Example 1.17** Let our alphabet consist of only three symbols $A$, $B$ and $C$ with probabilities of occurrence $P(A) = 0.5$, $P(B) = 0.25$ and $P(C) = 0.25$. We first divide the interval [0, 1) into three intervals proportional to their probabilities, as depicted in step 1 of Fig 1.16. Thus, the variable $A$ corresponds to [0, 0.5), the variable $B$ corresponds to [0.5, 0.75) and the variable $C$ corresponds to [0.75, 1.0). Note, that the lengths of these intervals are proportional to their probabilities. Next, suppose the input symbol stream is *B A C A* ... We first encode $B$. This is nothing but choosing the corresponding interval, i.e., [0.5,

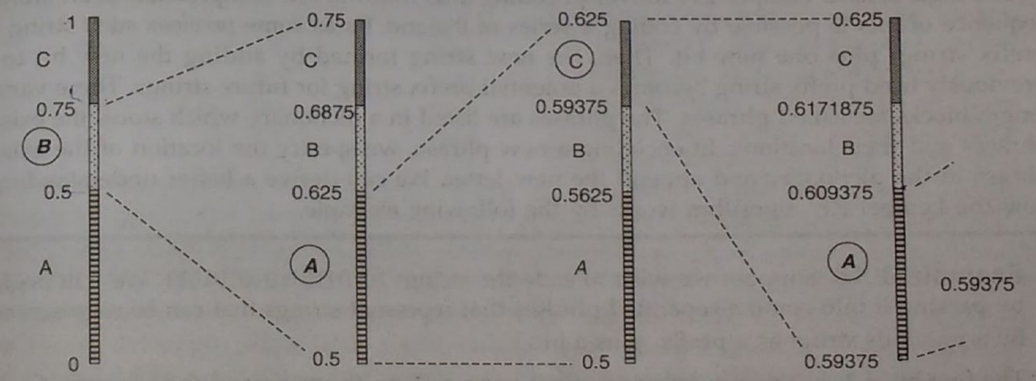


Fig. 1.16 An example to illustrate arithmetic coding of *B A C A*.

0.75). Now, this interval is again subdivided into three intervals, proportional to the probabilities of occurrence. So, for the second step (see Fig 1.16), the variable *A* corresponds to [0.5, 0.625), the variable *B* corresponds to [0.625, 0.6875) and the variable C corresponds to [0.6875, 1.0). Since the next symbol to arrive after *B* is *A*, we choose the interval corresponding to A, which is [0.5, 0.625). This is again subdivided to yield the interval [0.5, 0.5625) for *A*, the interval [0.5625, 0.59375) for *B*, and the interval [0.59375, 0.625) for *C*. Now we look at the next symbol to encode, which is *C*. This corresponds to the interval [0.59375, 0.625). Continuing this process, after encoding *A*, we are left with the interval [0.59375, 0.609375). The arithmetic code for *B A C A* is any number that lies within this interval. To complete this example, we can say that the arithmetic code for the sequence *B A C A* is 0.59375.

## 1.9 THE LEMPEL–ZIV ALGORITHM

Huffman coding requires symbol probabilities. But most real-life scenarios do not provide the symbol probabilities in advance (*i.e.*, the statistics of the source is unknown). In principle, it is possible to observe the output of the source for a long enough time period and estimate the symbol probabilities. However, this is impractical for real-time application. Also, Huffman coding is optimal for a DMS source where the occurrence of one symbol does not alter the probabilities of the subsequent symbols. Huffman coding is not the best choice for a source with memory. For example, consider the problem of compression of the the written text. We know that many letters occur in pairs or groups, like 'q-u', 't-h', 'i-n-g', etc. It might be more efficient to use the statistical inter-dependence of the letters in the alphabet along with their individual probabilities of occurrence. Such a scheme was proposed by Lempel and Ziv in 1977. Their source coding algorithm does not need the source statistics. It is a variable-to-fixed length source coding algorithm and belongs to the class of universal source coding algorithms.

The logic behind Lempel–Ziv universal coding is as follows. The compression of an arbitrary sequence of bits is possible by coding a series of 0's and 1's as some *previous* such string (the prefix string) plus one new bit. Then, the new string formed by adding the new bit to the previously used prefix string becomes a potential prefix string for future strings. These variable length blocks are called **phrases**. The phrases are listed in a dictionary which stores the existing phrases and their locations. In encoding a new phrase, we specify the location of the existing phrase in the dictionary and append the new letter. We can derive a better understanding of how the Lempel–Ziv algorithm works by the following example.

**Example 1.18** Suppose we wish to code the string: 101011011010101011. We will begin by **parsing** it into comma-separated phrases that represent strings that can be represented by a previous string as a prefix, plus a bit.

The first bit, a 1, has no predecessors, so, it has a null prefix string and the one extra bit is itself:



1, 01011011010101011.

The same goes for the 0 that follows since it can't be expressed in terms of the only existing prefix:

1, 0, 1011011010101011.

So far our dictionary contains the strings '1' and '0'. Next we encounter a 1, but it already exists in our dictionary. Hence we proceed further. The following 10 is obviously a combination of the prefix 1 and a 0, so we now have:

1, 0, 10, 11011010101011.

Continuing in this way we eventually parse the whole string as follows:

1, 0, 10, 11, 01, 101, 010, 1011.

Now, since we found 8 phrases, we will use a three bit code to label the null phrase and the first seven phrases for a total of 8 numbered phrases (with the ninth and last phrase we found being expressed in the others, hence not needing to be numbered). Next, we write the string in terms of the number of the prefix phrase plus the new bit needed to create the new phrase. We will use parentheses and commas to separate these at first, in order to aid our visualization of the process. The eight phrases can be described by:

(000,1)(000,0),(001,0),(001,1),(010,1),(011,1),(101,0),(110,1)

It can be read out as: (codeword at location 0,1), (codeword at location 0,0), (codeword at location 1,0), (codeword at location 1,1), (codeword at location 2,1), (codeword at location 3,1) ...

**Table 1.1** Dictionary for the Lempel–Ziv algorithm.

| *Dictionary Location* | *Dictionary content* | *Fixed Length Codeword* |
|---|---|---|
| 001 | 1 | 0001 |
| 010 | 0 | 0000 |
| 011 | 10 | 0010 |
| 100 | 11 | 0011 |
| 101 | 01 | 0101 |
| 110 | 101 | 0111 |
| 111 | 010 | 1010 |
| - | 1011 | 1101 |

Thus the coded version of the above string is:
000100000010001101010111101101.
The dictionary for this example is given in Table 1.1. In this case we have not obtained any compression, our coded string is actually longer. However, the larger the initial string, the more saving we get as we move along, because the prefixes that are quite large become representable as small numerical indices. In fact, Ziv proved that for long documents, the compression of the file approaches the optimum obtainable as determined by the information content of the document.

The next question is, what should be the length of the table. In practical application, regardless of the length of the table, it will eventually overflow. This problem can be solved by pre-deciding a large enough size of the dictionary. The encoder and decoder can update their dictionaries by periodically substituting the less used phrases from their dictionaries by the more frequently used ones. Lempel–Ziv algorithm is widely used in practice. The compress and

uncompress utilities of the UNIX operating system use a modified version of this algorithm. The standard algorithms for compressing binary files use code words of 12 bits and transmit 1 extra bit to indicate a new sequence. Using such a code, the Lempel-Ziv algorithm can compress transmissions of English text by about 55 percent, whereas the Huffman code compresses the transmission by only 43 percent.

In the following section, we will study another type of source coding scheme, particularly useful for facsimile transmission and image compression.

## 1.10 RUN-LENGTH ENCODING AND THE PCX FORMAT

Run-length Encoding, or RLE is a technique used to reduce the size of a repeating string of characters. This repeating string is called a **run**. Typically RLE encodes a run of symbols into two bytes, a count and a symbol. RLE can compress any type of data regardless of its information content, but the content of data to be compressed affects the compression ratio. RLE cannot achieve high compression ratios as compared to other compression methods, but it is easy to implement and is quick to execute. Run-length encoding is supported by most bitmap file formats such as TIFF, BMP and PCX.

---

**Example 1.19** Consider the following bit stream:

$$S = 11111111111111100000000000000000001111. \quad (1.51)$$

This can be represented as: fifteen 1's, nineteen 0's, four 1's, *i.e.*, (15,1), (19, 0), (4,1). Since the maximum number of repetitions is 19, which can be represented with 5 bits, we can encode the bit stream as (01111,1), (10011,0), (00100,1). The compression ratio in this case is 18:38 = 1:2.11.

---

RLE is highly suitable for FAX images of typical office documents. These two-colour images (black and white) are predominantly white. If we spatially sample these images for conversion into digital data, we find that many entire horizontal lines are entirely white (long runs of 0's). Furthermore, if a given pixel is black or white, the chances are very good that the next pixel will match. The code for fax machines is actually a combination of a run-length code and a Huffman code. A run-length code maps-run the lengths into code words, and the codebook is partitioned into two parts. The first part-contains symbols for runs of lengths that are a multiple of 64; the second part is made up of runs from 0 to 63 pixels. Any run-length would then be represented as a multiple of 64 plus some remainder. For example, a run of 205 pixels would be sent using the code word for a run of length 192 (3 x 64), plus the code word for a run of length 13. In this way the number of bits needed to represent the run is decreased significantly. In addition, certain runs that are known to have a higher probability of occurrence are encoded into code words of short length, further reducing the number of bits that are need to be transmitted. Using this type of encoding, typical compressions for facsimile transmission range between 4 to 1 and 8 to 1. Coupled to higher modem speeds, these compressions reduce the transmission time of a single page to less than a minute.



Run-length coding is also used for the compression of images in the PCX format. The PCX format was introduced as a part of the PC Paintbrush series of software for image painting and editing, sold by the ZSoft company. Today, the PCX format is actually an umbrella name for several possible image compression methods, and a means of identification that has been applied. We will restrict our attention here to only one of the methods, for 256 color images. We will restrict ourselves to that portion of the PCX data stream that actually contains the coded image, and not those parts that store the color palette and image information such as number of lines, pixels per line, file and the coding method.

The basic scheme is as follows. If a string of pixels are identical in color value, encode them as a special **flag byte** which contains the count followed by a byte with the value of the repeated pixel. If the pixel is not repeated, simply encode it as the byte itself. Such simple schemes can often become more complicated in practice. Consider that in the above scheme, if all 256 colors in a palette are used in an image, then, we need all 256 values of a byte to represent those colors. Hence if we are going to use just bytes as our basic code unit, we don't have any possible unused byte values that can be used as a flag/count byte. On the other hand, if we use two bytes for every coded pixel, so as to leave room for the flag/count combinations, we might double the size of the pathological images instead of compressing them.

The compromise in the PCX format is based on the belief of its designers than many user created drawings (which was the primary intended output of their software) would not use all 256 colors. So, they optimised their compression scheme for the case of up to 192 colors only. Images with more colors will probably also get good compression, just not quite as good, with this scheme.

---

**Example 1.20** PCX compression encodes single occurrences of color (that is, a pixel that is not part of a run of the same color) 0 through 191 simply as the binary byte representation of exactly that numerical value. Consider the Table 1.2.

**Table 1.2** Example of PCX encoding

| *Pixel color value* | *Hex code* | *Binary code* |
|---|---|---|
| 0 | 00 | 00000000 |
| 1 | 01 | 00000001 |
| 2 | 02 | 00000010 |
| 3 | 03 | 00000011 |
| : | : | : |
| 190 | BE | 10111110 |
| 191 | BF | 10111111 |

For the color 192 (and all the colors higher than 192), the codeword is equal to one byte in which the two most significant bits (MSBs) are both set to a 1. We will use these codewords to signify a flag and count byte. If the two MSBs are equal to one, we will say

that they have flagged a count. The remaining 6 bits in the flag/count byte will be interpreted as a 6 bit binary number for the count (from 0 to 63). This byte is then followed by the byte which represents the color. In fact, if we have a run of pixels of one of the colors with palette code even over 191, we can still code the run easily since the top two bits are not reserved in this second color code byte of a run-coding byte pair.

If a run of pixels exceeds 63 in length, we simply use this code for the first 63 pixels in the run, and that runs additional codes of that pixel until we exhaust all pixels in the run. The next question is: how do we code those remaining colors in a nearly full palette image when there is no run? We still code these as a run by simply setting the run length to 1. That means, for the case of at most 64 colors which appear as single pixels in the image and not part of runs, we expand the data by a factor of two. Luckily this rarely happens.

In the next section, we will study coding for analog sources. Recall that we ideally need infinite number of bits to accurately represent an analog source. Anything fewer will only be an approximate representation. We can choose to use fewer and fewer bits for representation at the cost of a poorer approximation of the original signal (rule of thumb: there is no free lunch). Thus, quantisation of the amplitudes of the sampled signals results in data compression. We would like to study the distortion introduced when the samples from the information source are quantised.

## 1.11 RATE DISTORTION FUNCTION

Although we live in an analog world, most of the communication takes place in the digital form. Since most natural sources (e.g. speech, video etc.) are analog, they are first sampled, quantised and then processed. However, the representation of an arbitrary real number requires an infinite number of bits. Thus, a finite representation of a continuous random variable can never be perfect.

Consider an analog message waveform $x(t)$ which is a sample waveform of a stochastic process $X(t)$. Assuming $X(t)$ is a bandlimited, stationary process, it can be represented by a sequence of uniform samples taken at the Nyquist rate. These samples are quantised in amplitude and encoded as a sequence of binary digits. A simple encoding strategy can be to define $L$ levels and encode every sample using

$$R = \log_2 L \text{ bits if } L \text{ is a power of 2, or}$$
$$R = \lfloor \log_2 L \rfloor + 1 \text{ bits if } L \text{ is not a power of 2.} \quad (1.52)$$

If all levels are not equally probable we may use entropy coding for a more efficient representation. In order to represent the analog waveform more accurately, we need more number of levels, which would imply more number of bits per sample. Theoretically we need infinite bits per sample to perfectly represent an analog source. Quantisation of amplitude results in data compression at the cost of signal distortion. It is a form of lossy data compression. Distortion implies some measure of difference between the actual source samples $\{x_k\}$ and the corresponding quantised value $\{\tilde{x}_k\}$.



**Definition 1.14** The **Squared Error Distortion** is defined as

$$d(x_k, \tilde{x}_k) = (x_k - \tilde{x}_k)^2 . \quad (1.53)$$

In general a distortion measure may be represented as

$$d(x_k, \tilde{x}_k) = |x_k - \tilde{x}_k|^p . \quad (1.54)$$

Consider a sequence of $n$ samples $X_n$, and the corresponding $n$ quantized values $\tilde{X}_n$. Let $d(x_k, \tilde{x}_k)$ be the distortion measure per sample (letter). Then the distortion measure between the original sequence and the sequence of quantized values will simply be the average over the $n$ source output samples i.e.,

$$d(X_n, \tilde{X}_n) = \frac{1}{n}\sum_{k=1}^{n} d(x_k, \tilde{x}_k) . \quad (1.55)$$

We observe that the source is a random process, hence $X_n$ and consequently $d(X_n, \tilde{X}_n)$ are random variables. We now define the distortion as follows.

**Definition 1.15** The **Distortion** between a sequence of $n$ samples $X_n$, and their corresponding $n$ quantised values $\tilde{X}_n$ is defined as

$$D = E\,[d(X_n, \tilde{X}_n)] = \frac{1}{n}\sum_{k=1}^{n} E[d(x_k, \tilde{x}_k)] = E[d(x_k, \tilde{x}_k)] . \quad (1.56)$$

It has been assumed here that the random process is stationary. Next, let a memoryless source have a continuous output $X$ and the quantised output alphabet $\tilde{X}$. Let the probability density function of this continuous amplitude be $p(x)$ and per letter distortion measure be $d(x, \tilde{x})$, where $x \in X$ and $\tilde{x} \in \tilde{X}$. We next introduce the rate distortion function, which gives us the minimum number of bits per sample required to represent the source output symbols given a prespecified allowable distortion.

**Definition 1.16** The minimum rate (in bits/source output) required to represent the output $X$ of the memoryless source with a distortion less than or equal to $D$ is called the **Rate Distortion Function** $R(D)$, defined as

$$R(D) = \min_{p(\tilde{x}|x):E[d(X,\tilde{X})]\le D} I(X;\tilde{X}) , \quad (1.57)$$

where $I(X; \tilde{X})$ is the average mutual information between $X$ and $\tilde{X}$.

We will now state (without proof) two theorems related to the rate distortion function.

For $f(x) = x^2$, i.e., the mean square value of the distortion, equations (1.64) simplify to

$$x_k = \frac{1}{2}(\tilde{x}_k + \tilde{x}_{k+1}) \qquad k = 1,2,....L-1$$
$$\int_{x_{k-1}}^{x_k} (\tilde{x}_k - x)\, p(x)dx = 0 \qquad k = 1,2,....L \tag{1.65}$$

The non uniform quantisers are optimised with respect to the distortion. However, each quantised sample is represented by *equal* number of bits (say, $R$ bits/sample). It is possible to have a more efficient variable length coding. The discrete source outputs that result from quantisation can be characterised by a set of probabilities $p_k$. These probabilities can then be used to design efficient variable length codes (source coding). In order to compare the performance of different non-uniform quantisers, we first fix the distortion, $D$, and then compare the average number of bits required per sample.

**Example 1.22** Consider an eight-level quantiser for a Gaussian random variable. This problem was first solved by Max in 1960. The random variable has zero mean and variance equal to unity. For a mean square error minimisation, the values $x_k$ and $\tilde{x}_k$ are listed in Table 1.3.

**Table 1.3** Optimum quantization and Huffman coding

| *Level, k* | $x_k$ | $\tilde{x}_k$ | $P(x_k)$ | *Huffman Code* |
|---|---|---|---|---|
| 1 | – 1.748 | – 2.152 | 0.040 | 0010 |
| 2 | – 1.050 | – 1.344 | 0.107 | 011 |
| 3 | – 0.500 | – 0.756 | 0.162 | 010 |
| 4 | 0 | – 0.245 | 0.191 | 10 |
| 5 | 0.500 | 0.245 | 0.191 | 11 |
| 6 | 1.050 | 0.756 | 0.162 | 001 |
| 7 | 1.748 | 1.344 | 0.107 | 0000 |
| 8 | ∞ | 2.152 | 0.040 | 0011 |

For these values, $D = 0.0345$ which equals –14.62 dB.

The number of bits/sample for this optimum 8-level quantiser is $R = 3$. On performing Huffman coding, the average number of bits per sample required is $R_H = 2.88$ bits/sample. The theoretical limit is $H(X) = 2.82$ bits/sample.

## 1.13 ENTROPY RATE OF A STOCHASTIC PROCESS

A simple extension of the source coding theorem tells us that $nH(X)$ bits are sufficient, on an average, to describe $n$ independent and identically distributed random variables, each with



entropy $H(X)$. But, in the real world, we do encounter random variables that are dependent. What if the random variables form a stationary process?

**Definition 1.18** A stochastic process is said to be **Stationary** if the joint distribution of any subset of the sequence of random variables is invariant with respect to shifts in time index, i.e.,

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = P(X_{1+m} = x_1, X_{2+m} = x_2, \ldots X_{n+m} = x_n). \tag{1.66}$$

for every shift $m$ and for all $x_1, x_2, \ldots x_n \in X$.

**Definition 1.19** A discrete stochastic process $X_1, X_2, \ldots$ is said to be a **Markov Chain** or a **Markov Process** if, for $n = 1, 2 \ldots$

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_{n+1} = x_{n+1} \mid X_n = x_n), \tag{1.67}$$

for all $x_1, x_2, \ldots, x_n, x_{n+1} \in X$.

The probability density function of a Markov process can be written as

$$p(x_1, x_2, \ldots, x_n) = p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_2) \ldots, p(x_n \mid x_{n-1}). \tag{1.68}$$

If we have a sequence of $n$ random variables, it is interesting to explore how the entropy of the sequence grows with $n$. Entropy rate is used to define this rate of growth.

**Definition 1.20** The **Entropy Rate** of a stochastic process $X$ is given by

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2 \ldots . X_n), \tag{1.69}$$

provided the limit exists.

**Example 1.23** Let $X_1, X_2, X_3, \ldots$ be independent and identically distributed random variables. In this case the entropy rate would be

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n) = \lim_{n\to\infty} \frac{nH(X_1)}{n} = H(X_1).$$

However, if $X_1, X_2, X_3, \ldots$ form a sequence of independent but *not* identically distributed random variables, then

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i).$$

Thus, the entropy rate is

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} H(X_i).$$

- *Run length Encoding, or RLE is a technique used to reduce the size of a repeating string of characters. This repeating string is called a run. Run length encoding is supported by many bitmap file formats such as TIFF, BMP and PCX.*
- *Distortion implies some measure of difference between the actual source samples $\{x_i\}$ and the corresponding quantised value $\{\tilde{x}_i\}$. The squared error distortion is given by*

$$d(x_i, \tilde{x}_i) = (x_i - \tilde{x}_i)^2.$$

*In general, a distortion measure may be represented as*

$$d(x_i, \tilde{x}_i) = |x_i - \tilde{x}_i|^r.$$

- *The minimum rate (in bits/source output) required to represent the output $X$ of a memoryless source with a distortion less than or equal to $D$ is called the rate distortion function $R(D)$, defined as*

$$R(D) = \min_{p(\tilde{x}|x):\, E[d(X,\tilde{X})] \le D} I(X;\tilde{X}),$$

*where $I(X; \tilde{X})$ is the average mutual information between $X$ and $\tilde{X}$.*

- *The distortion resulting due to the quantisation can be expressed as*

$$D = \int_{-\infty}^{\infty} f(\tilde{x} - x) p(x)\, dx,$$

*where $f(\tilde{x} - x)$ is the desired function of the error. An optimum quantiser is one that minimizes $D$ by optimally selecting the output levels and the corresponding input range of each output level. The resulting optimum quantiser is called the Lloyd–Max quantiser.*

- *A discrete stochastic process $X_1, X_2, \ldots$ is said to be a Markov chain or a Markov process if for $n = 1, 2 \ldots$*

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_{n+1} = x_{n+1} \mid X_n = x_n)$$

*for all*

$$x_1, x_2, \ldots x_n, x_{n+1} \in X.$$

- *The entropy rate of a stochastic process $X$ is given by*

$$H(X) = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n)$$

*provided the limit exists.*

- *For stationary Markov chain, the entropy rate is given by*

$$H(X) = \lim_{n \to \infty} H(X_n \mid X_{n-1}, \ldots, X_1) = \lim_{n \to \infty} H(X_n \mid X_{n-1}) = H(X_2 \mid X_1).$$

- *Quantisation and source coding techniques (Huffman coding, arithmetic coding and run length coding) are used in the JPEG standard for image compression.*



> Obstacles are those frightful things you see when you take your eyes off your goal.
> - Henry Ford (1863–1947)

## PROBLEMS

1.1 Consider a DMS with source probabilities {0.30, 0.25, 0.20, 0.15, 0.10}. Find the source entropy, $H(X)$.

1.2 Prove that the entropy for a discrete source is a maximum when the output symbols are equally probable.

1.3 Prove the inequality $\ln x \le x - 1$. Plot the curves $y_1 = \ln x$ and $y_2 = x - 1$ to demonstrate the validity of this inequality.

1.4 Show that $I(X; Y) \ge 0$. Under what condition does the equality hold?

1.5 A source, $X$, has an infinitely large set of outputs with probability of occurrence given by $P(x_i) = 2^{-i}$, $i = 1, 2, 3, \ldots$. What is the average self information, $H(X)$, of this source?

1.6 Consider another geometrically distributed random variable $X$ with $P(x_i) = p(1-p)^{i-1}$, $i = 1, 2, 3, \ldots$. What is the average self-information, $H(X)$, of this source?

1.7 Let $X$ denote the number of tosses required for a coin until the first tail appears.
- (i) Find the entropy, $H_f(X)$ if the coin is fair.
- (ii) Next, assume the coin to be unfair with the $p$ being the probability of getting a tail. Find the entropy, $H_u(X)$.

1.8 Consider a code that satisfies the suffix condition: No codeword is a suffix of any other codeword.
- (i) Is this a uniquely decodable code? Why?
- (ii) Can you relate the minimum average length of Suffix codes to the average length of the Huffman code for the same random variable? Explain.

1.9 The relative entropy or Kullback–Leibler distance between two probability mass functions $p(x)$ and $q(x)$ is defined as

$$D(p \,\|\, q) = \sum_{x \in X} p(x) \log \left( \frac{p(x)}{q(x)} \right) \tag{1.76}$$

- (i) Show that $D(p \,\|\, q)$ is non-negative.
- (ii) Show that the Kullback–Leibler distance does not follow the other two properties of a distance, i.e., symmetry and the triangle inequality.
- (iii) Show that $I(X; Y) = D(p(x,y) \,\|\, p(x)p(y))$. This would elegantly prove that mutual information is non negative.

1.10 Suppose a source produces *independent* symbols from the alphabet $\{a_1, a_2, a_3\}$, with probabilities $p_1 = 0.4999999$, $p_2 = 0.4999999$, and $p_3 = 0.0000002$.
- (i) Compute the entropy, $H(X)$, of this source.
- (ii) Find an optimal code for this source, and compute its expected codeword length.

(iii) Find an optimal code for the second extension of this source (i.e., for blocks of two symbols), and compute its expected codeword length, and the expected codeword length divided by two.

(iv) Prove that in order to compress to within 1% of the entropy by encoding blocks of size $N$ from this source, $N$ will have to be at least 5. Note that if you use (1.45) you get a very loose upper bound.

1.11 We would like to encode a sequence of symbols that come from an alphabet with $d + 3$ symbols. We want to encode symbols $a_1$, $a_2$, and $a_3$ using codewords that are three bits long. We want to encode symbols $a_4, a_5, \ldots, a_{d+3}$ using codewords that are eight bits long. What is the maximum value of $d$ for which this will be possible, if the code must be uniquely decodable?

1.12 Consider an integer values random variable, X, given by

$$P(X=n)=\frac{1}{An\log^2 n}$$

where $A=\sum_{n=2}^{\infty}\frac{1}{n\log^2 n}$ and $n = 2, 3, \ldots, \infty$. Find the entropy, $H(X)$.

1.13 Calculate the differential entropy, $H(X)$, of the uniformly distributed random variable $X$ with the pdf,

$$p(x)=\begin{cases}a^{-1} & (0\le x\le a)\\ 0 & (otherwise)\end{cases}. \qquad (1.77)$$

Plot the differential entropy, $H(X)$, versus the parameter $a$ $(0.1 < a < 10)$. Comment on the result.

1.14 Consider a DMS with source probabilities {0.35, 0.25, 0.20, 0.15, 0.05}.
- (i) Determine the Huffman code for this source.
- (ii) Determine the average length $\bar{R}$ of the codewords.
- (iii) What is the efficiency $\eta$ of the code?

1.15 Again consider a DMS with source probabilities {0.35, 0.25, 0.20, 0.15, 0.05}
- (i) Determine the ternary Huffman code for this source using symbols 1, 2 and 3.
- (ii) Determine the average length $\bar{R}$ of the codewords and compare it with the binary case.

1.16 Consider a DMS with source probabilities {0.20, 0.20, 0.15, 0.15, 0.10, 0.10, 0.05, 0.05}.
- (i) Determine an efficient fixed length code for the source.
- (ii) Determine the Huffman code for this source.
- (iii) Compare the two codes and comment.

1.17 A DMS has three output symbols with probabilities {0.5, 0.4, 0.1}.
- (i) Determine the Huffman code for this source and find the efficiency $\eta$.
- (ii) Determine the Huffman code for this source taking *two* symbols at a time and find the efficiency $\eta$.
- (iii) Determine the Huffman code for this source taking *three* symbols at a time and find the efficiency $\eta$.



1.18 For a source with entropy $H(X)$, prove that the entropy of a $B$-symbol block is $BH(X)$.

1.19 Let $X$ and $Y$ be random variables that take on values $x_1, x_2, \ldots, x_r$ and $y_1, y_2, \ldots, y_s$ respectively. Let $Z = X + Y$.
- (i) Show that $H(Z|X) = H(Y|X)$.
- (ii) If X and Y are independent, then argue that $H(Y) \le H(Z)$ and $H(X) \le H(Z)$. Comment on this observation.
- (iii) Under what condition will $H(Z) = H(X) + H(Y)$?

1.20 Determine the Lempel-Ziv code for the following bit stream 01001111100101000001 010101100110000. Recover the original sequence from the encoded stream.

1.21 Consider Lempel–Ziv encoding for quaternary data (symbols: 0, 1, 2, 3).
- (i) Encode the following quaternary data: 1 3 3 0 0 2 0 2 1 1 1 3 0 0 0 0 2 2 1 2 2 2 3 3. What is the compression ratio obtained, where the compression ratio is defined as the number of bits after encoding divided by the number of bits before encoding.
- (ii) Next, represent each quaternary symbol by its binary equivalent ($0 \to 00$, $1 \to 01$, $2 \to 10$, $3 \to 11$). Now perform Lempel–Ziv encoding on the binary data and obtain the compression ratio.
- (iii) Compare the results of (i) and (ii). Comment.

1.22 Consider a bit stream being generated by source A. The probabilities of occurrence of 'runs' in the bit stream is given by

$$p_r=\begin{cases}2^{-r} & \text{for } 1\le r<n\\ 2^{-(n-1)} & \text{for } r=n\end{cases}, \qquad (1.78)$$

where $r$ is the length of the run and $n$ is the maximum run-length possible. Note that the probabilities correspond to the run-length, irrespective of whether it's a run of 1's or 0's.
- (i) Design a run length code and find out the compression it will provide.
- (ii) Suppose you want to encode the runs using Huffman code. Generate the code for this bit stream. What is the compression provided by this code?

1.23 We next want to try a new scheme for run-length coding. We repeat the run-length encoding process again and again until no more compression is possible. Using this procedure, if we have to encode a run of, say $n$ 1's, what will be the maximum possible (best case) compression?

1.24 Find the rate distortion function $R(D) = \min I(X; \tilde{X})$ for Bernoulli distributed X, with $p = 0.5$, where the distortion is given by

$$d(x,\tilde{x})=\begin{cases}0 & x=\tilde{x}\\ 1 & x=1,\ \tilde{x}=0\\ \infty & x=0,\ \tilde{x}=1\end{cases}, \qquad (1.79)$$

1.25 Consider a source $X$ uniformly distributed on the set {1, 2, …, $m$}. Find the rate distortion function for this source with Hamming distortion defined as

$$d(x,\tilde{x})=\begin{cases}0, & x=\tilde{x}\\ 1, & x\ne\tilde{x}\end{cases}. \qquad (1.80)$$

## COMPUTER PROBLEMS

1.26 Write a program that performs Huffman coding, given the source probabilities. It should generate the code and give the coding efficiency.

1.27 Modify the above program so that it can group together $n$ source symbols and then generate the Huffman code. Plot the coding efficiency $\eta$ versus $n$ for the following source symbol probabilities: {0.55, 0.25, 0.20}. For what value of $n$ does the efficiency become better than 0.9999? Repeat the exercise for following source symbol probabilities {0.45, 0.25, 0.15, 0.10, 0.05}.

1.28 Write a program that executes the Lempel–Ziv algorithm. The input to the program can be the English alphabets. It should convert the alphabets to their ASCII code and then perform the compression routine. It should output the compression achieved. Using this program, find out the compression achieved for the following strings of letters:

(i) The Lempel–Ziv algorithm can compress the English text by about fifty–five percent.

(ii) The cat cannot sit on the canopy of the car.

1.29 Write a program that performs run length encoding (RLE) on a sequence of bits and gives the coded output along with the compression ratio. What is the output of the program if the following sequence is fed into it:

11000000001111000001111111111111111111000000110000000.

Now feed back the *encoded output* to the program, *i.e.*, perform the RLE two times on the original sequence of bits. What do you observe? Comment on your observation.

1.30 Suppose for a certain type of communication we only use three letters A, B and C. The probability of occurrences for these are {0.5 0.3, 0.2}. However, the use of these letters is not independent. The probability of occurrences of the bigrams are P(AB) = 0.25, P(BC) = 0.05, P(CA) = 0.35, P(BA) = 0.15, P(CB) = 0.05, P(AC) = 0.15. Let us call the Huffman code which takes into consideration the bigrams as Pseudo Markov Huffman (PMH) code.

(i) Design a PMH code for this communication scenario which takes into consideration the bigrams as well. Write a computer program that can take these probabilities as input and generate the PMH code.

(ii) Write a program that performs arithmetic coding for this case and compare with the result of the PMH code.

1.31 Write a program that takes in a $2^n$ level gray scale image ($n$ bits per pixel) and performs the following operations:

(i) Breaks it up in to 8 by 8 pixel blocks.

(ii) Performs DCT on *each* of the 8 by 8 blocks.

(iii) Quantizes the DCT coefficients by retaining only the $m$ most significant bits (MSB), where $m \leq n$.

(iv) Performs the zig-zag coding followed by run length coding.

(v) Performs Huffman coding on the bit stream obtained above (think of a reasonable way of calculating the symbol probabilities).



(vi) Calculates the compression ratio.

(vii) Performs the decompression (*i.e.*, the inverse operation of the steps (v) back to (i)).

Perform image compression using this program for different values of $m$. Up to what value of $m$ is there no perceptible difference in the original image and the compressed image?

## 2.2 CHANNEL MODELS

We have already come across the simplest of the channel models, the **Binary Symmetric Channel** (BSC), in the previous chapter. If the modulator employs binary waveforms and the detector makes hard decisions, then the channel may be viewed as one in which a binary bit stream enters at the transmitting end and another bit stream comes out at the receiving end. This is depicted in Fig. 2.2.

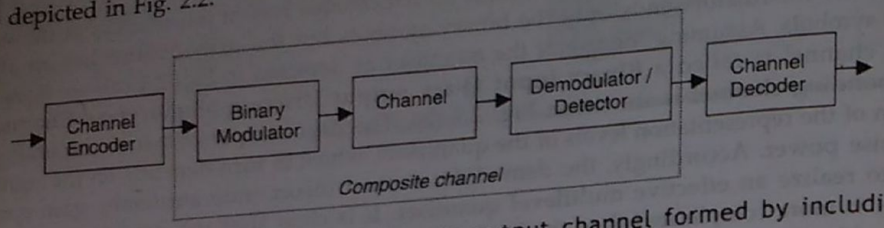


**Fig. 2.2** A composite discrete-input, discrete-output channel formed by including the modulator and demodulator/detector.

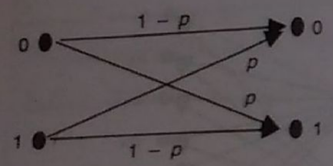


**Fig. 2.3** A binary symmetric channel (BSC).

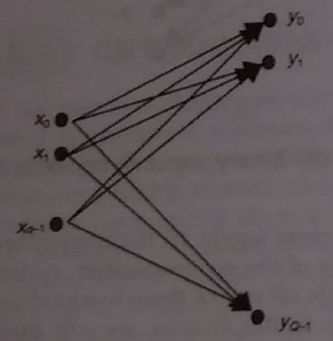


**Fig. 2.4** A discrete memoryless channel (DMC) with $q$-ary input and $Q$-ary output.

The composite **Discrete-input, Discrete-output channel** is characterised by the set $X = \{0,1\}$ of possible inputs, the set $Y = \{0,1\}$ of possible outputs and a set of conditional probabilities that relate the possible outputs to the possible inputs. Let the noise in the channel cause independent errors in the transmitted binary sequence with average probability of error $p$

$$P(Y = 0 \mid X = 1) = P(Y = 1 \mid X = 0) = p,$$
$$P(Y = 1 \mid X = 1) = P(Y = 0 \mid X = 0) = 1 - p. \qquad (2.1)$$

A binary symmetric channel is shown in Fig. 2.3.

The BSC is a special case of a general, discrete input, discrete output channel. Let the input to the channel be $q$-ary symbols, i.e., $X = \{x_0, x_1, .., x_{q-1}\}$ and the output of the detector at the receiving end of the channel consist of $Q$-ary symbols, i.e., $Y = \{y_0, y_1, .., y_{Q-1}\}$. We assume that the channel and the modulation is memoryless. The inputs and the outputs can then be related by a set of $qQ$ conditional probabilities

$$P(Y = y_i \mid X = x_j) = P(y_i \mid x_j), \qquad (2.2)$$

where $i = 0,1, \ldots Q - 1$ and $j = 0,1, \ldots q - 1$. This channel is known as a **Discrete Memoryless Channel** (DMC) and is depicted in Fig. 2.4.



**Definition 2.1** The conditional probability $P(y_i \mid x_j)$ is defined as the **Channel Transition Probability** and is denoted by $p_{ji}$.

**Definition 2.2** The conditional probabilities $\{P(y_i \mid x_j)\}$ that characterise a DMC can be arranged in the matrix form $P = [p_{ji}]$. $P$ is called the **Probability Transition Matrix** for the channel.

So far we have discussed a single channel with discrete inputs and discrete outputs. It is also possible to realise multiple channels between the transmitter and receiver. Such channels can be readily realised in wireless communication scenarios by using multiple antennas at the transmitter and receiver. The four obvious combinations are listed below.

(i) ***Single Input Single Output*** (SISO): This refers to the familiar wireless configuration with a single antenna both at the transmitter and receiver.

(ii) ***Single Input Multiple Output*** (SIMO): This refers to the configuration with a single antenna at the transmitter and multiple antennas at the receiver.

(iii) ***Multiple Input Single Output*** (MISO): This refers to the configuration with multiple antennas at the transmitter but only a single antenna at the receiver.

(iv) ***Multiple Input Multiple Output*** (MIMO): This refers to the most general configuration with multiple antennas both at the transmitter and receiver.

Consider a MIMO system with $M_T$ transmit antennas and $M_R$ receive antennas. Let us denote the impulse response between the $j^{\text{th}}$ ($j = 1, 2, \ldots M_T$) transmit antenna and the $i^{\text{th}}$ ($i = 1, 2, \ldots M_R$) receiving antenna by $h_{i,j}(\tau, t)$. Note that here we are considering the waveform channel and the modulator/demodulator is not a part of the channel. The MIMO channel can be represented using a $M_R \times M_T$ matrix as follows:

$$\boldsymbol{H}(\tau, t) = \begin{bmatrix} h_{1,1}(\tau,t) & h_{1,2}(\tau,t) & .... & h_{1,M_T}(\tau,t) \\ h_{2,1}(\tau,t) & h_{2,2}(\tau,t) & .... & h_{2,M_T}(\tau,t) \\ \vdots & \vdots & .... & \vdots \\ h_{M_R,1}(\tau,t) & h_{M_R,2}(\tau,t) & .... & h_{M_R,M_T}(\tau,t) \end{bmatrix}. \qquad (2.3)$$

The variable $\tau$ is used to capture the time varying nature of the channel. If a signal $s_j(t)$ is transmitted from the $j^{\text{th}}$ transmit antenna, the signal received at the $i^{\text{th}}$ receive antenna is given by

$$y_i(t) = \sum_{j=1}^{M_T} h_{i,j}(\tau,t) s_j(t), \; i = 1,2, \ldots, M_R. \qquad (2.4)$$

The input output relation of a MIMO channel can be expressed succinctly in matrix notation as

$$\boldsymbol{y}(t) = \boldsymbol{H}(\tau, t)\, \boldsymbol{s}(t), \qquad (2.5)$$

where, $\boldsymbol{s}(t) = [s_1(t) \;\; s_2(t) \;\ldots\; S_{M_T}(t)]^T$ and , $\boldsymbol{y}(t) = [y_1(t) \;\; y_2(t) \;\ldots\; y_{M_T}(t)]^T$.

## 2.2 CHANNEL MODELS

We have already come across the simplest of the channel models, the **Binary Symmetric Channel** (BSC), in the previous chapter. If the modulator employs binary waveforms and the detector makes hard decisions, then the channel may be viewed as one in which a binary bit stream enters at the transmitting end and another bit stream comes out at the receiving end. This is depicted in Fig. 2.2.

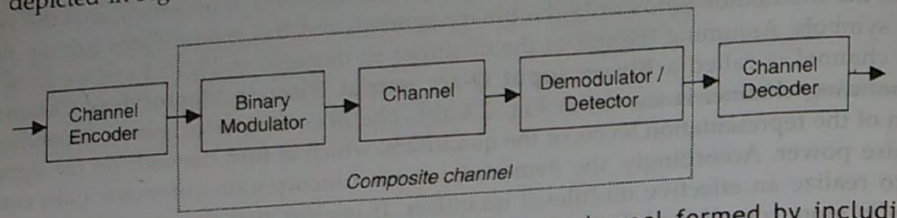


**Fig. 2.2** A composite discrete-input, discrete-output channel formed by including the modulator and demodulator/detector.

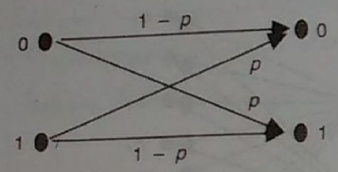


**Fig. 2.3** A binary symmetric channel (BSC).

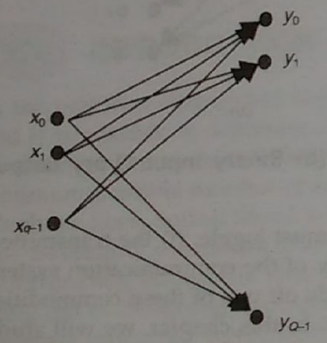


**Fig. 2.4** A discrete memoryless channel (DMC) with $q$-ary input and $Q$-ary output.

The composite **Discrete-input, Discrete-output channel** is characterised by the set $X = \{0,1\}$ of possible inputs, the set $Y = \{0,1\}$ of possible outputs and a set of conditional probabilities that relate the possible outputs to the possible inputs. Let the noise in the channel cause independent errors in the transmitted binary sequence with average probability of error $p$

$$P(Y = 0 \mid X = 1) = P(Y = 1 \mid X = 0) = p,$$
$$P(Y = 1 \mid X = 1) = P(Y = 0 \mid X = 0) = 1 - p. \qquad (2.1)$$

A binary symmetric channel is shown in Fig. 2.3.

The BSC is a special case of a general, discrete input, discrete output channel. Let the input to the channel be $q$-ary symbols, i.e., $X = \{x_0, x_1, .., x_{q-1}\}$ and the output of the detector at the receiving end of the channel consist of $Q$-ary symbols, i.e., $Y = \{y_0, y_1, .., y_{Q-1}\}$. We assume that the channel and the modulation is memoryless. The inputs and the outputs can then be related by a set of $qQ$ conditional probabilities

$$P(Y = y_i \mid X = x_j) = P(y_i \mid x_j), \qquad (2.2)$$

where $i = 0,1, \ldots Q - 1$ and $j = 0,1, \ldots q - 1$. This channel is known as a **Discrete Memoryless Channel** (DMC) and is depicted in Fig. 2.4.



**Definition 2.1** The conditional probability $P(y_i \mid x_j)$ is defined as the **Channel Transition Probability** and is denoted by $p_{ji}$.

**Definition 2.2** The conditional probabilities $\{P(y_i \mid x_j)\}$ that characterise a DMC can be arranged in the matrix form $P = [p_{ji}]$. $P$ is called the **Probability Transition Matrix** for the channel.

So far we have discussed a single channel with discrete inputs and discrete outputs. It is also possible to realise multiple channels between the transmitter and receiver. Such channels can be readily realised in wireless communication scenarios by using multiple antennas at the transmitter and receiver. The four obvious combinations are listed below.

(i) ***Single Input Single Output*** (SISO): This refers to the familiar wireless configuration with a single antenna both at the transmitter and receiver.

(ii) ***Single Input Multiple Output*** (SIMO): This refers to the configuration with a single antenna at the transmitter and multiple antennas at the receiver.

(iii) ***Multiple Input Single Output*** (MISO): This refers to the configuration with multiple antennas at the transmitter but only a single antenna at the receiver.

(iv) ***Multiple Input Multiple Output*** (MIMO): This refers to the most general configuration with multiple antennas both at the transmitter and receiver.

Consider a MIMO system with $M_T$ transmit antennas and $M_R$ receive antennas. Let us denote the impulse response between the $j^{th}$ ($j = 1, 2, \ldots M_T$) transmit antenna and the $i^{th}$ ($i = 1, 2, \ldots M_R$) receiving antenna by $h_{i,j}(\tau, t)$. Note that here we are considering the waveform channel and the modulator/demodulator is not a part of the channel. The MIMO channel can be represented using a $M_R \times M_T$ matrix as follows:

$$\boldsymbol{H}(\tau, t) = \begin{bmatrix} h_{1,1}(\tau,t) & h_{1,2}(\tau,t) & .... & h_{1,M_T}(\tau,t) \\ h_{2,1}(\tau,t) & h_{2,2}(\tau,t) & .... & h_{2,M_T}(\tau,t) \\ \vdots & \vdots & .... & \vdots \\ h_{M_R,1}(\tau,t) & h_{M_R,2}(\tau,t) & .... & h_{M_R,M_T}(\tau,t) \end{bmatrix}. \qquad (2.3)$$

The variable $\tau$ is used to capture the time varying nature of the channel. If a signal $s_j(t)$ is transmitted from the $j^{th}$ transmit antenna, the signal received at the $i^{th}$ receive antenna is given by

$$y_i(t) = \sum_{j=1}^{M_T} h_{i,j}(\tau,t) s_j(t), \; i = 1,2, \ldots, M_R. \qquad (2.4)$$

The input output relation of a MIMO channel can be expressed succinctly in matrix notation as

$$\boldsymbol{y}(t) = \boldsymbol{H}(\tau, t)\, \boldsymbol{s}(t), \qquad (2.5)$$

where, $\boldsymbol{s}(t) = [s_1(t) \;\; s_2(t) \;\ldots\; S_{M_T}(t)]^T$ and , $\boldsymbol{y}(t) = [y_1(t) \;\; y_2(t) \;\ldots\; y_{M_T}(t)]^T$.

Each link between a pair of transmitter and receiver can be independently represented as a discrete memoryless channel.

## 2.3 CHANNEL CAPACITY

Consider a DMC having an input alphabet $X = \{x_0, x_1, \ldots, x_{q-1}\}$ and an output alphabet $y = \{y_0, y_1, \ldots, y_{r-1}\}$. Let us denote the set of channel transition probabilities by $P(y_i | x_j)$. The average mutual information provided by the output $Y$ about the input $X$ is given by (see Chapter 1, Section 1.3)

$$I(X;Y) = \sum_{j=0}^{q-1} \sum_{i=0}^{r-1} P(x_j) P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)} \quad (2.6)$$

The channel transition probabilities $P(y_i | x_j)$ are determined by the channel characteristics (primarily the noise in the channel). However, the input symbol probabilities $P(x_j)$ are within the control of the discrete channel encoder. The value of the average mutual information, $I(X;Y)$, maximised over the set of input symbol probabilities $P(x_j)$ is a quantity that depends only on the channel transition probabilities $P(y_i | x_j)$ (hence only on the characteristics of the channel). This quantity is called the **Capacity of the Channel**.

**Definition 2.3** The **Capacity** of a discrete memoryless channel (DMC) is defined as the maximum average mutual information in any single use of the channel, where the maximisation is over all possible input probabilities. That is,

$$C = \max_{P(x_j)} I(X;Y)$$

$$= \max_{P(x_j)} \sum_{j=0}^{q-1} \sum_{i=0}^{r-1} P(x_j) P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)} . \quad (2.7)$$

The maximization of $I(X;Y)$ is performed under the constraints

$$P(x_j) \geq 0, \text{ and } \sum_{j=0}^{q-1} P(x_j) = 1 .$$

The units of channel capacity is bits per channel use (provided the base of the logarithm is 2).

**Example 2.1** Consider a BSC with channel transition probabilities

$$P(0|1) = p = P(1|0).$$

By symmetry, the capacity, $C = \max_{P(x_j)} I(X;Y)$, is achieved for $p = 0.5$. From (2.7) we obtain



the capacity of a BSC as

$$C = 1 + p\log_2 p + (1-p)\log_2(1-p).$$

Let us define the entropy function

$$H(p) = -p\log_2 p - (1-p)\log_2(1-p).$$

Hence we can rewrite the capacity of a binary symmetric channel as

$$C = 1 - H(p).$$

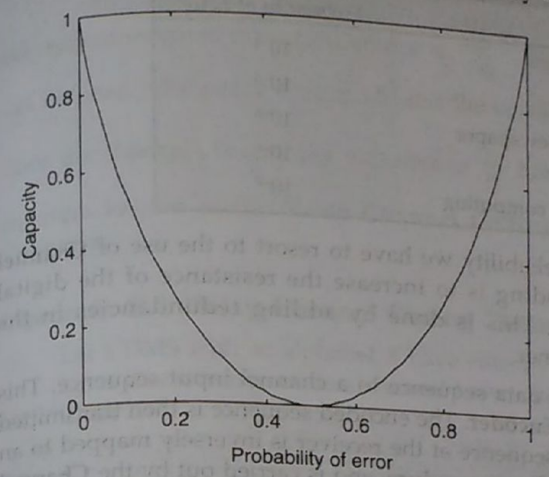


**Fig. 2.5** The capacity of a BSC.

The plot of the capacity versus $p$ is given in Fig. 2.5. From the plot we make the following observations:

(i) For $p = 0$ (i.e., noise-free channel), the capacity is 1 bit/use, as expected. Each time we use the channel, we can successfully transmit 1 bit of information.

(ii) For $p = 0.5$, the channel capacity is 0, i.e., observing the output gives no information about the input. It is equivalent to the case when the channel is broken. We might as well, discard the channel and toss a fair coin in order to estimate what was transmitted.

(iii) For $0.5 < p < 1$, the capacity increases with increasing $p$. In this case we simply reverse the positions of 1 and 0 at the output of the BSC.

(iv) For $p = 1$ (i.e., every bit get flipped by the channel), the capacity is again 1 bit/use, as expected. In this case, one simply flips the bit at the output of the receiver so as to undo the effect of the channel.

(v) Since $p$ is a monotonically decreasing function of SNR, the capacity of a BSC is a monotonically increasing function of SNR.

Having developed the notion of capacity of a channel, we shall now try to relate it to a reliable communication over the channel. So far, we have only talked about bits that can be sent over a channel each time it is used (bits/use). But, what is the number of bits that can be sent per second (bits/sec)? To answer this question we introduce the concept of **Channel Coding**.

## 2.4 CHANNEL CODING

All real-life channels are affected by noise. Noise causes discrepancies (errors) between the input and the output data sequences of a digital communication system. For a typical noisy channel, the probability of bit error may be as high as $10^{-2}$. This means that, on an average, 1

bit out of every 100 bits that are transmitted over this channel gets flipped. For most applications, this level of reliability is far from adequate. Different applications require different *levels of reliability* (which is a component of the quality of service). Table 2.1 lists the typical acceptable bit error rates for various applications.

**Table 2.1** Acceptable bit error rates for various applications.

| *Application* | *Probability of Error* |
|---|---|
| Speech telephony | $10^{-4}$ |
| Voice band data | $10^{-6}$ |
| Electronic mail, Electronic newspaper | $10^{-6}$ |
| Internet access | $10^{-6}$ |
| Video telephony, High speed computing | $10^{-7}$ |

In order to achieve such high levels of reliability we have to resort to the use of **Channel Coding**. The basic objective of channel coding is to increase the resistance of the digital communication system to channel noise. This is done by adding redundancies in the transmitted data stream in a controlled manner.

In channel coding, we map the incoming data sequence to a channel input sequence. This encoding procedure is done by the **Channel Encoder**. The encoded sequence is then transmitted over the noisy channel. The channel output sequence at the receiver is inversely mapped to an output data sequence. This is called the decoding procedure, and is carried out by the **Channel Decoder**. Both the encoder and the decoder are under the designer's control.

As already mentioned, the encoder introduces redundancy in a prescribed manner. The decoder exploits this redundancy so as to reconstruct the original source sequence as accurately as possible. Thus, channel coding makes it possible to carry out reliable communication over unreliable (noisy) channels. Channel coding is also referred to as **Error Control Coding**. It is interesting to note here that the source coder reduces redundancy to improve efficiency, where as, the channel coder adds redundancy, in a controlled manner, to improve reliability.

We first look at a class of channel codes called *block codes*. In this class of codes, the incoming message sequence is first sub-divided into sequential blocks, each of length $k$ bits. Each $k$-bit long information block is mapped into an $n$-bit block by the channel coder, where $n > k$. This means that for every $k$ bits of information, $(n - k)$ redundant bits are added. The ratio

$$r = \frac{k}{n} \tag{2.8}$$

is called the **Code Rate**. Code rate of any coding scheme is, naturally, less than unity. A small code rate implies that more and more bits per block are the redundant bits corresponding to a higher coding overhead. This may reduce the effect of noise, but will also reduce the communication rate as we will end up transmitting more of the redundant bits and fewer information bits. The question, before us is whether there exists a coding scheme such that the



probability that the message bit will be in error is arbitrarily small and yet the coding rate is not too small? The answer is yes, and was first provided by Shannon in his second theorem regarding the channel capacity. We will study this shortly.

Let us now introduce the concept of time in our discussion. We wish to look at questions like how many bits per second can we send over a given noisy channel with arbitrarily low bit error rates? Suppose the discrete memoryless source has the source alphabet $X$ and entropy $H(X)$ bits per source symbol. Let the source generate a symbol every $T_s$ seconds. Then the average information rate of the source is $\frac{H(X)}{T_s}$ bits per second. Let us assume that the channel can be used once every $T_c$ seconds and the capacity of the channel is $C$ bits per channel use. Then the channel capacity per unit time is $\frac{C}{T_c}$ bits per second. We now state Shannon's second theorem known as the **Noisy Channel Coding Theorem** or simply the **Channel Coding Theorem**.

### Theorem 2.1 Noisy Channel Coding Theorem

(i) Let a DMS with an alphabet $X$ have entropy $H(X)$ and produce symbols every $T_s$ seconds. Let a discrete memoryless channel have a capacity $C$ and be used once every $T_c$ seconds. Then if

$$\frac{H(X)}{T_s} \leq \frac{C}{T_c}, \tag{2.9}$$

there exists a coding scheme for which the source output can be transmitted over the noisy channel and be reconstructed with an arbitrarily low probability of error.

(ii) Conversely if

$$\frac{H(X)}{T_s} > \frac{C}{T_c}, \tag{2.10}$$

it is not possible to transmit information over the channel and reconstruct it with an arbitrarily small probability of error.

The parameter $\frac{C}{T_c}$ is called the **Critical Rate**.

The channel coding theorem is a very important result in information theory. The theorem specifies the channel capacity, $C$, as a fundamental limit on the rate at which reliable communication can be carried out over an unreliable (noisy) DMS channel. It should be noted that the channel coding theorem tells us about the existence of some codes that can achieve reliable communications in a noisy environment. Unfortunately, it does not give us the recipe to construct these codes. Therefore, channel coding is still an active area of research as search for better and better codes is still going on. From the next chapter onwards we shall study some good channel codes.

**Example 2.2** Consider a DMS source that emits equally likely binary symbols ($p = 0.5$) once every $T_s$ seconds. The entropy for this binary source is

$$H(p) = -p\log_2 p - (1-p)\log_2(1-p) = 1 \text{ bit.}$$

The information rate of this source is

$$\frac{H(X)}{T_s} = \frac{1}{T_s} = \text{ bits/second.}$$

Suppose we wish to transmit the source symbols over a noisy channel. The source sequence is applied to a channel coder with code rate $r$. This channel coder uses the channel once every $T_c$ seconds to send the coded sequence. We want to have a reliable communication (the probability of error as small as desired). From the channel coding theorem, if

$$\frac{1}{T_s} \le \frac{C}{T_c}, \tag{2.11}$$

we can make the probability of error as small as desired by a suitable choice of a channel coding scheme, and hence have reliable communication. We note that the ratio is equal to the code rate of the coder. Therefore,

$$r = \frac{T_c}{T_s}, \tag{2.12}$$

Hence the condition for reliable communication can be rewritten as

$$r \le C. \tag{2.13}$$

Thus, for a BSC one can find a suitable channel coding scheme with a code rate, $r \le C$, which will ensure reliable communication regardless of how noisy the channel is. Of course, we can state that at least one such code exists, but finding that code may not be such a trivial job. As we shall see later, the level of noise in the channel will manifest itself by limiting the channel capacity, and hence the code rate.

**Example 2.3** Consider a binary symmetric channel (BSC) with a transition probability $p = 10^{-2}$. Such error rates are typical of wireless channels. We saw in example 2.1 that for a BSC the capacity is given by

$$C = 1 + p\log_2 p + (1-p)\log_2(1-p)$$

By plugging in the value of $p = 10^{-2}$ we obtain the channel capacity $C = 0.919$. From the previous example we can conclude that there exists at least one coding scheme, with the code rate $r \le 0.919$ which will guarantee us a (non zero) probability of error that is as small as desired.



**Example 2.4** Consider the **repetition code** in which each message bit is simply repeated $n$ times, where $n$ is an odd integer. For example, for $n = 3$, we have the mapping scheme

$$0 \to 000;\ 1 \to 111.$$

Similarly, for $n = 5$ we have the mapping scheme

$$0 \to 00000;\ 1 \to 11111.$$

Note that the code rate of the repetition code with blocklength $n$ is

$$r = \frac{1}{n}. \tag{2.14}$$

The decoding strategy is as follows: If in a block of $n$ received bits the number of 0's exceeds the number of 1's, decide in favour of 0 and *vice-versa*. This is also known as **Majority Decoding**. This also answers the question why should $n$ be an odd integer for repetition codes.

Let $n = 2m + 1$, where $m$ is a positive integer. This decoding strategy will make an error if more than $m$ bits are in error, because in that case even if a 0 is encoded and sent, there will be more number of 1's in the received word. Let us assume that the *a priori* probabilities of 1 and 0 are equal. Then, the average probability of error is given by

$$P_e = \sum_{i=m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}, \tag{2.15}$$

where $p$ is the channel transition probability. The average probability of error for repetition codes for different code rates is given in Table 2.2.

**Table 2.2** Average probability of error for repetition codes.

| *Code rate, r* | *Average probability of error, $P_e$* |
|---|---|
| 1 | $10^{-2}$ |
| 1/3 | $3\times10^{-4}$ |
| 1/5 | $10^{-6}$ |
| 1/7 | $4\times10^{-7}$ |
| 1/9 | $10^{-8}$ |
| 1/11 | $5\times10^{-10}$ |

From the Table we see that as the code rate decreases, there is a steep fall in the average probability of error. The decrease in the $P_e$ is much more rapid than the decrease in the code rate, $r$. However, for repetition codes, the code rate tends to zero if we want smaller and smaller $P_e$. Thus the repetition code exchanges code rate for message reliability. But the channel coding theorem states that the code rate need not tend to zero in order to

obtain an arbitrarily low probability of error. The theorem merely requires the code rate $r$ to be less than the channel capacity, $C$. So there must exist some code (other than the repetition code) with code rate $r \approx 0.9$ which can achieve arbitrarily low probability of error. Such a coding scheme will add just 1 parity bit to 9 information bits (or, maybe, add 10 extra bits to 90 information bits) and give us as small a $P_e$ as desired (say, $10^{-20}$). The hard part is finding such a code.

## 2.5 INFORMATION CAPACITY THEOREM

So far we have studied limits on the maximum rate at which information can be sent over a channel reliably in terms of the channel capacity. In this section, we will formulate the information capacity theorem for band limited, power limited Gaussian channels. Consider a zero mean, stationary random process $X(t)$ that is band limited to $W$ hertz. Let $X_k$, $k = 1, 2,..., K$, denote the continuous random variables obtained by uniform sampling of the process $X(t)$ at the Nyquist rate of $2W$ samples per second. These symbols are transmitted over a noisy channel which is also band limited to $W$ Hertz. The channel output is corrupted by AWGN of zero mean and power spectral density (psd) $N_0/2$. Because of the channel, the noise is band limited to $W$ Hertz. Let $Y_k$, $k = 1, 2,..., K$, denote the samples of the received signal. Therefore,

$$Y_k = X_k + N_k, \; k = 1, 2,..., K, \tag{2.16}$$

where $N_k$ is the noise sample with zero mean and variance $\sigma^2 = N_0W$. It is assumed that $Y_k$, $k = 1, 2,..., K$, are statistically independent. Since the transmitter is usually power limited, let us put a constraint on the *average* power in $X_k$:

$$E(X_k^2) = P, \; k = 1, 2,..., K. \tag{2.17}$$

The information capacity of this band limited, power limited channel is the maximum of the mutual information between the channel input $X_k$ and the channel output $Y_k$. The maximisation has to be done over all distributions on the input $X_k$ that satisfy the power constraint of equation (2.17). Thus, the information capacity of the channel (same as the channel capacity) is given by,

$$C = \max_{f_{X_k}(x)} \{I(X;Y) \mid E[X_k^2] = P\}, \tag{2.18}$$

where $f_{X_k}(x)$ is the probability density function of $X_k$.

Now, from the previous chapter, equation (1.32), we have,

$$I(X_k;Y_k) = h(Y_k) - h(Y_k \mid X_k). \tag{2.19}$$

Note that $X_k$ and $N_k$ are independent random variables. Therefore, the conditional differential entropy of $Y_k$ given $X_k$ is equal to the differential entropy of $N_k$. Intuitively, this is because given $X_k$ the uncertainty arising in $Y_k$ is purely due to $N_k$. That is,

$$h(Y_k \mid X_k) = h(N_k). \tag{2.20}$$



Hence we can write equation (2.19) as

$$I(X_k; Y_k) = h(Y_k) - h(N_k). \tag{2.21}$$

Since $h(N_k)$ is independent of $X_k$, maximising $I(X_k; Y_k)$ translates to maximising $h(Y_k)$. It can be shown that in order for $h(Y_k)$ to be maximum, $Y_k$ has to be a Gaussian random variable (see problem 2.10). If we assume $Y_k$ to be Gaussian, and $N_k$ is Gaussian by definition, then $X_k$ is also Gaussian. This is because the sum (or difference) of two Gaussian random variables is also Gaussian. Thus, in order to maximise the mutual information between the channel input $X_k$ and the channel output $Y_k$, the transmitted signal should be Gaussian. Therefore we can rewrite (2.18) as

$$C = I(X;Y) \mid E[X_k^2] = P \text{ and } X_k \text{ is Gaussian.} \tag{2.22}$$

We know that if two independent Gaussian random variables are added, the variance of the resulting Gaussian random variable is the sum of the variances. Therefore, the variance of the received sample $Y_k$ equals $P + N_0W$. It can be shown that the differential entropy of a Gaussian random variable with variance $\sigma^2$ is $\frac{1}{2}\log_2(2\pi e\sigma^2)$ (see problem 2.10). Therefore,

$$h(Y_k) = \frac{1}{2}\log_2[2\pi e(P + N_0W)], \tag{2.23}$$

and

$$h(N_k) = \frac{1}{2}\log_2[2\pi e(N_0W)]. \tag{2.24}$$

Substituting these values of differential entropy for $Y_k$ and $N_k$ we get

$$C = \frac{1}{2}\log_2\left(1 + \frac{P}{N_0W}\right) \text{ bits per channel use.} \tag{2.25}$$

We are transmitting $2W$ samples per second, i.e., the channel is being used $2W$ times in one second. Therefore, the information capacity can be expressed as

$$C = W \log_2\left(1 + \frac{P}{N_0W}\right) \text{ bits per second.} \tag{2.26}$$

This basic formula for the capacity of the band limited, AWGN waveform channel with a band limited and average power limited input was first derived by Shannon in 1948. It is known as the Shannon's third theorem, the **Information Capacity Theorem**.

**Theorem 2.2 (Information Capacity Theorem)** The information capacity of a continuous channel of bandwidth $W$ Hertz, perturbed by additive white Gaussian noise of power spectral density $N_0/2$ and limited in bandwidth to $W$, is given by

obtain an arbitrarily low probability of error. The theorem merely requires the code rate $r$ to be less than the channel capacity, $C$. So there must exist some code (other than the repetition code) with code rate $r = 0.9$ which can achieve arbitrarily low probability of error. Such a coding scheme will add just 1 parity bit to 9 information bits (or, maybe, add 10 extra bits to 90 information bits) and give us as small a $P_e$ as desired (say, $10^{-20}$). The hard part is finding such a code.

## 2.5 INFORMATION CAPACITY THEOREM

So far we have studied limits on the maximum rate at which information can be sent over a channel reliably in terms of the channel capacity. In this section, we will formulate the information capacity theorem for band limited, power limited Gaussian channels. Consider a zero mean, stationary random process $X(t)$ that is band limited to $W$ hertz. Let $X_k$, $k = 1, 2, \ldots, K$, denote the continuous random variables obtained by uniform sampling of the process $X(t)$ at the Nyquist rate of $2W$ samples per second. These symbols are transmitted over a noisy channel which is also band limited to $W$ Hertz. The channel output is corrupted by AWGN of zero mean and power spectral density (psd) $N_0/2$. Because of the channel, the noise is band limited to $W$ Hertz. Let $Y_k$, $k = 1, 2, \ldots, K$, denote the samples of the received signal. Therefore,

$$Y_k = X_k + N_k, \; k = 1, 2, \ldots, K, \tag{2.16}$$

where $N_k$ is the noise sample with zero mean and variance $\sigma^2 = N_0W$. It is assumed that $Y_k$, $k = 1, 2, \ldots, K$, are statistically independent. Since the transmitter is usually power limited, let us put a constraint on the *average* power in $X_k$:

$$E(X_k^2) = P, \; k = 1, 2, \ldots, K. \tag{2.17}$$

The information capacity of this band limited, power limited channel is the maximum of the mutual information between the channel input $X_k$ and the channel output $Y_k$. The maximisation has to be done over all distributions on the input $X_k$ that satisfy the power constraint of equation (2.17). Thus, the information capacity of the channel (same as the channel capacity) is given by,

$$C = \max_{f_{X_k}(x)} \{I(X;Y) \mid E[X_k^2] = P\}, \tag{2.18}$$

where $f_{X_k}(x)$ is the probability density function of $X_k$.

Now, from the previous chapter, equation (1.32), we have,

$$I(X_k; Y_k) = h(Y_k) - h(Y_k \mid X_k). \tag{2.19}$$

Note that $X_k$ and $N_k$ are independent random variables. Therefore, the conditional differential entropy of $Y_k$ given $X_k$ is equal to the differential entropy of $N_k$. Intuitively, this is because given $X_k$ the uncertainty arising in $Y_k$ is purely due to $N_k$. That is,

$$h(Y_k \mid X_k) = h(N_k). \tag{2.20}$$



Hence we can write equation (2.19) as

$$I(X_k; Y_k) = h(Y_k) - h(N_k). \tag{2.21}$$

Since $h(N_k)$ is independent of $X_k$, maximising $I(X_k; Y_k)$ translates to maximising $h(Y_k)$. It can be shown that in order for $h(Y_k)$ to be maximum, $Y_k$ has to be a Gaussian random variable (see problem 2.10). If we assume $Y_k$ to be Gaussian, and $N_k$ is Gaussian by definition, then $X_k$ is also Gaussian. This is because the sum (or difference) of two Gaussian random variables is also Gaussian. Thus, in order to maximise the mutual information between the channel input $X_k$ and the channel output $Y_k$, the transmitted signal should be Gaussian. Therefore we can rewrite (2.18) as

$$C = I(X;Y) \mid E[X_k^2] = P \text{ and } X_k \text{ is Gaussian.} \tag{2.22}$$

We know that if two independent Gaussian random variables are added, the variance of the resulting Gaussian random variable is the sum of the variances. Therefore, the variance of the received sample $Y_k$ equals $P + N_0W$. It can be shown that the differential entropy of a Gaussian random variable with variance $\sigma^2$ is $\frac{1}{2}\log_2(2\pi e\sigma^2)$ (see problem 2.10). Therefore,

$$h(Y_k) = \frac{1}{2}\log_2[2\pi e(P + N_0W)], \tag{2.23}$$

and

$$h(N_k) = \frac{1}{2}\log_2[2\pi e(N_0W)]. \tag{2.24}$$

Substituting these values of differential entropy for $Y_k$ and $N_k$ we get

$$C = \frac{1}{2}\log_2\left(1 + \frac{P}{N_0W}\right) \text{ bits per channel use.} \tag{2.25}$$

We are transmitting $2W$ samples per second, i.e., the channel is being used $2W$ times in one second. Therefore, the information capacity can be expressed as

$$C = W \log_2\left(1 + \frac{P}{N_0W}\right) \text{ bits per second.} \tag{2.26}$$

This basic formula for the capacity of the band limited, AWGN waveform channel with a band limited and average power limited input was first derived by Shannon in 1948. It is known as the Shannon's third theorem, the **Information Capacity Theorem**.

**Theorem 2.2 (Information Capacity Theorem)** The information capacity of a continuous channel of bandwidth $W$ Hertz, perturbed by additive white Gaussian noise of power spectral density $N_0/2$ and limited in bandwidth to $W$, is given by

The concept of cutoff rate was also developed by Shannon, but was later used by Wozencraft, Jacobs and Kennedy as a design parameter for communication systems. Jordan used the concept of cutoff rate to design coded waveforms for M-ary orthogonal signals with coherent and non-coherent detection. Cutoff rates have been widely used as a design criterion for various different channels, including fading channels encountered in wireless communications.

## SUMMARY

- The conditional probability $P(y_i|x_j)$ is called the channel transition probability and is denoted by $p_{ji}$. The conditional probabilities $\{P(y_i|x_j)\}$ that characterise a DMC can be arranged in the matrix form $\mathbf{P} = [p_{ji}]$. $\mathbf{P}$ is known as the probability transition matrix for the channel.
- The capacity of a discrete memoryless channel (DMC) is defined as the maximum average mutual information in any single use of the channel, where the maximisation is over all possible input probabilities. That is,

$$C = \max_{P(x_j)} I(X;Y) = \max_{P(x_j)} \sum_{j=0}^{q-1}\sum_{i=0}^{r-1} P(x_j)P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)}.$$

- The basic objective of channel coding is to increase the resistance of the digital communication system to channel noise. This is done by adding redundancies in the transmitted data stream in a controlled manner. Channel coding is also referred to as error control coding.
- The ratio, $r = \frac{k}{n}$, is called the code rate. Code rate of any coding scheme is always less than unity.
- Let a DMS with an alphabet $X$ have entropy $H(X)$ and produce symbols every $T_s$ seconds. Let a DMC have capacity $C$ and be used once every $T_c$ seconds. Then, if $\frac{H(X)}{T_s} \le \frac{C}{T_c}$, there exists a coding scheme for which the source output can be transmitted over the noisy channel and be reconstructed with an arbitrarily low probability of error. This is the Channel Coding Theorem or the Noisy Channel Coding Theorem.
- For $\frac{H(X)}{T_s} > \frac{C}{T_c}$, it is not possible to transmit information over the channel and reconstruct it with an arbitrarily small probability of error. The parameter $\frac{C}{T_c}$ is called the Critical Rate.
- The information capacity of a SISO channel can be expressed as $C = W \log_2\left(1+\frac{P}{N_0 W}\right)$ bits per second. This is the basic formula for the capacity of the band limited, AWGN waveform channel with a band limited and average power limited input. This is the crux of the Information Capacity Theorem. This theorem is also called the Channel Capacity Theorem.



- The capacity of a frequency flat deterministic MIMO channel is given by

$$C = \max_{\mathrm{Tr}(R_{ss})=M_T} W \log_2 \det\left(I_{M_R} + \frac{E_s}{M_T N_0} HR_{ss}H^H\right) \text{ bits per second}$$

where W is the bandwidth. The condition $\mathrm{Tr}(\mathbf{R}_{ss}) = M_T$ constrains the total average energy transmitted over a symbol period. Here the assumption is that the channel is known to the receiver.
- If the channel is unknown to the transmitter, the capacity of the MIMO channel is given by

$$C = W\sum_{i=1}^{r} \log_2\left(1+\frac{E_s}{M_T N_0}\lambda_i\right) \text{ bits per second,}$$

where $r$ is the rank of the channel and $\lambda_i$ $(i = 1, 2, ..., r)$ are the positive eigenvalues of $\mathbf{HH}^H$.
- If the channel is known to the transmitter, the different scalar data pipes may be accessed individually through processing at the transmitter and receiver. The optimal energy for each data pipe is found iteratively using the Waterpouring Algorithm.
- The cutoff rate $R_0$ is given by

$$R_0 = \log_2 \frac{2}{1+e^{\frac{-E}{N_0}}} = 1 - \log_2\left(1+e^{\frac{-E}{N_0}}\right).$$

The cutoff rate has the units of bits/dimension. Note that $0 \le R_0 \le 1$. The average error probability in terms of the cutoff rate can be written as $\overline{P}_e < 2^{-n(R_0-R_c)}$. For $R_c < R_0$ the average probability of error $\overline{P}_e \to 0$ as $n \to \infty$.

> Mathematics is like checkers in being suitable for the young, not too difficult, amusing, and without peril to the state.
>
> Plato (ca. 429–347 BC)

## PROBLEMS

2.1 Consider the binary channel shown in Fig. 2.10. Let the a priori probabilities of sending the binary symbol be $p_0$ and $p_1$, where $p_0 + p_1 = 1$. Find the *a posteriori* probabilities

$P(X = 0 | Y = 0)$ and $P(X = 1 | Y = 1)$.

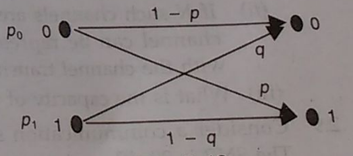

Fig. 2.10

2.2 Find the capacity of the binary *erasure* channel shown in Fig. 2.11, where $p_0$ and $p_1$ are the *a priori* probabilities.

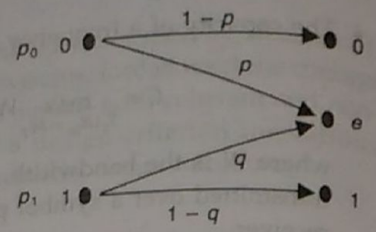

Fig. 2.11

2.3 Consider the channels A, B and the cascaded channel AB shown in Fig. 2.12.
- (i) Find $C_A$ the capacity of channel *A*.
- (ii) Find $C_B$ the capacity of channel *B*.
- (iii) Next, cascade the two channels and determine the combined capacity $C_{AB}$.
- (iv) Explain the relation between $C_A$, $C_B$ and $C_{AB}$.

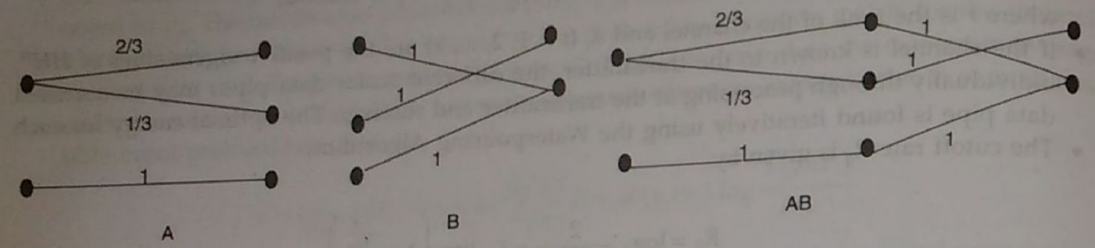

Fig. 2.12

2.4 Find the capacity of the channel shown in Fig. 2.13.

2.5 (i) A telephone channel has a bandwidth of 3000 Hz and the SNR = 20 dB. Determine the channel capacity.
- (ii) If the SNR is increased to 25 dB, determine the increased capacity.

2.6 Determine the channel capacity of the channel shown in Fig. 2.14.

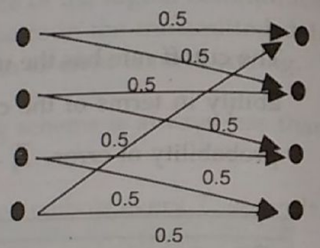

Fig. 2.13

2.7 Suppose a TV displays 30 frames/second. There are approximately 2 × 105 pixels per frame, each pixel requiring 16 bits for colour display. Assuming an SNR of 25 dB calculate the bandwidth required to support the transmission of the TV video signal (use the information capacity theorem).

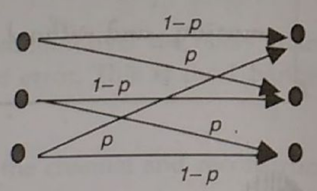

Fig. 2.14

2.8 Consider the *Z* channel shown Fig. 2.15.
- (i) Find the input probabilities that result in capacity.
- (ii) If *N* such channels are cascaded, show that the combined channel can be represented by an equivalent *Z* channel with the channel transition probability $(1 - p)^N$.
- (iii) What is the capacity of the combined channel as $N \to \infty$ ?

2.9 Consider a communication system using antipodal signaling. The SNR is 20 dB.

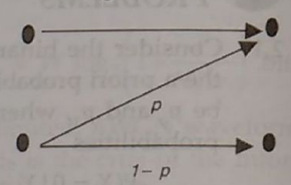

Fig. 2.15



- (i) Find the cutoff rate, $R_0$.
- (ii) We want to design a code which results in an average probability of error, $P_e < 10^{-6}$. What is the best code rate one can achieve?
- (iii) What will be the dimensionality, *n*, of this code?
- (iv) Repeat the earlier parts (i), (ii) and (iii) for an SNR = 5dB. Compare the results.

2.10 (i) Prove that for a finite variance $\sigma^2$, the Gaussian random variable has the largest differential entropy attainable by any random variable.

(ii) Show that this entropy is given by $\frac{1}{2}\log_2(2\pi e\sigma^2)$.

## COMPUTER PROBLEMS

2.11 Write a computer program that takes in the channel transition probability matrix and computes the capacity of the channel.

2.12 Plot the operating points on the bandwidth efficiency diagram for *M*-PSK, *M* = 2, 4, 8, 16 and 32, and the probabilities of error: (a) $P_e = 10^{-6}$ and (b) $P_e = 10^{-8}$.

2.13 Write a program that implements the binary repetition code of rate $1/n$, where *n* is an odd integer. Develop a decoder for the repetition code. Test the performance of this coding scheme over a BSC with the channel transition probability, *p*. Generalize the program for a repetition code of rate $1/n$ over *GF* (*q*). Plot the residual Bit Error Rate (BER) versus *p* and *q* (make a 3-*D* mesh plot).

2.14 Write a program that can generate Capacity versus SNR plots for $M_T$ transmit antennas and $M_R$ receive antennas.
- (i) Generate plots for the combinations $(M_T, M_R) = (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (3, 1)$.
- (ii) Compare the capacity of a MISO and a SIMO channel. Comment.
- (iii) If we have a total number of 4 antennas, which is the best solution: (1, 3), (2, 2) or (3, 1)? Justify.