Acknowledgements

I would like to thank the Department of Electrical Engineering at the Indian Institute of Technology (IIT), Delhi for providing a stimulating academic environment that inspired this book. In particular, I would like to thank Prof. S.C. Dutta Roy, Prof. Surendra Prasad, Prof. H.M. Gupta, Prof. V.K. Jain, Prof. Vinod Chandra, Prof. Santanu Chaudhury, Prof. S.D. Joshi, Prof. Sheel Aditya, Prof. Devi Chadha, Prof. D. Nagchoudri, Prof. G.S. Visweswaran, Prof. R. K. Patney, Prof. V. C. Prasad, Prof. S. S. Jamuar and Prof. R. K. P. Bhatt. I am also thankful to Dr. Subrat Kar, Dr. Ranjan K. Mallik and Dr. Shankar Prakriya for friendly discussions. I have been fortunate to have several batches of excellent students whose feedback have helped me improve the contents of this book. Many of the problems given at the end of the chapters have been tested either as assignment problems or examination problems.

My heartfelt gratitude is due to Prof. Bernard D. Steinberg, University of Pennsylvania, who has been my guide, mentor, friend and also my Ph. D thesis advisor. I am also grateful to Prof. Avraham Freedman, Tel Aviv University, for his support and suggestions as and when sought by me. I would like to thank Prof. B. Sundar Rajan of the Electrical Communication Engineering group at the Indian Institute of Science, Bangalore, with whom I had a preliminary discussion about writing this book.

I wish to acknowledge valuable feedback on this initial manuscript from Prof. Ravi Motwani, IIT Kanpur, Prof. A.K. Chaturvedi, IIT Kanpur, Prof. N. Kumaravel, Anna University, Prof. V. Maleswara Rao, College of Engineering, GITAM, Visakhapatnam, Prof. M. Chandrasekaran, Government College of Engineering, Salem and Prof. Vikram Gadre, IIT Mumbai.

I am indebted to my parents, for their love and moral support throughout my life. I am also grateful to my grandparents for their blessings, and to my younger brother, Shantanu, for the infinite discussions on finite topics.

Finally, I would like to thank my wife and best friend, Aloka, who encouraged me at every stage of writing this book. Her constructive suggestions and balanced criticism have been instrumental in making the book more readable and palatable. It was her infinite patience, unending support, understanding and the sense of humour that were critical in transforming my dream into this book.

RANJAN BOSE
New Delhi

Contents

Preface IX
Acknowledgements XII

Part I
Information Theory and Source Coding

1. Source Coding 3
1.1 Introduction to Information Theory *3*
1.2 Uncertainty And Information *4*
1.3 Average Mutual Information And Entropy *11*
1.4 Information Measures For Continuous Random Variables *14*
1.5 Source Coding Theorem *15*
1.6 Huffman Coding *21*
1.7 The Lempel-Ziv Algorithm *28*
1.8 Run Length Encoding and the PCX Format *30*
1.9 Rate Distortion Function *33*
1.10 Optimum Quantizer Design *36*
1.11 Introduction to Image Compression *37*
1.12 The Jpeg Standard for Lossless Compression *38*
1.13 The Jpeg Standard for Lossy Compression *39*
1.14 Concluding Remarks *41*
Summary *42*
Problems *44*
Computer Problems *46*

2. Channel Capacity and Coding 47
2.1 Introduction *47*
2.2 Channel Models *48*
2.3 Channel Capacity *50*
2.4 Channel Coding *52*
2.5 Information Capacity Theorem *56*

2.6 The Shannon Limit *59*
2.7 Random Selection of Codes *61*
2.8 Concluding Remarks *67*
Summary *68*
Problems *69*
Computer Problems *71*

Part II
Error Control Coding
(Channel Coding)

3. Linear Block Codes for Error Correction **75**
3.1 Introduction to Error Correcting Codes *75*
3.2 Basic Definitions *77*
3.3 Matrix Description of Linear Block Codes *81*
3.4 Equivalent Codes *82*
3.5 Parity Check Matrix *85*
3.6 Decoding of a Linear Block Code *87*
3.7 Syndrome Decoding *94*
3.8 Error Probability after Coding (Probability of Error Correction) *95*
3.9 Perfect Codes *97*
3.10 Hamming Codes *100*
3.11 Optimal Linear Codes *102*
3.12 Maximum Distance Separable (MDS) Codes *102*
3.13 Concluding Remarks *102*
Summary *103*
Problems *105*
Computer Problems *106*

4. Cyclic Codes **108**
4.1 Introduction to Cyclic Codes *108*
4.2 Polynomials *109*
4.3 The Division Algorithm for Polynomials *110*
4.4 A Method for Generating Cyclic Codes *115*
4.5 Matrix Description of Cyclic Codes *119*
4.6 Burst Error Correction *121*

4.7 Fire Codes *123*
4.8 Golay Codes *124*
4.9 Cyclic Redundancy Check (CRC) Codes *125*
4.10 Circuit Implementation of Cyclic Codes *128*
4.11 Concluding Remarks *132*
Summary *132*
Problems *134*
Computer Problems *135*

5. Bose-Chaudhuri Hocquenghem (BCH) Codes **136**
5.1 Introduction to BCH Codes *136*
5.2 Primitive Elements *137*
5.3 Minimal Polynomials *139*
5.4 Generator Polynomials in Terms of Minimal Polynomials *141*
5.5 Some Examples of BCH Codes *143*
5.6 Decoding of BCH Codes *147*
5.7 Reed-Solomon Codes *150*
5.8 Implementation of Reed-Solomon Encoders and Decoders *153*
5.9 Nested Codes *153*
5.10 Concluding Remarks *155*
Summary *156*
Problems *157*
Computer Problems *158*

6. Convolutional Codes **159**
6.1 Introduction to Convolutional Codes *159*
6.2 Tree Codes and Trellis Codes *160*
6.3 Polynomial Description of Convolutional Codes (Analytical Representation) *165*
6.4 Distance Notions for Convolutional Codes *170*
6.5 The Generating Function *173*
6.6 Matrix Description of Convolutional Codes *176*
6.7 Viterbi Decoding of Convolutional Codes *178*
6.8 Distance Bounds for Convolutional Codes *185*
6.9 Performance Bounds *187*
6.10 Known Good Convolutional Codes *188*
6.11 Turbo Codes *190*

6.12 Turbo Decoding *192*
6.13 Concluding Remarks *198*
Summary *199*
Problems *201*
Computer Problems *203*

7. Trellis Coded Modulation **206**
7.1 Introduction to TCM *206*
7.2 The Concept of Coded Modulation *207*
7.3 Mapping by Set Partitioning *212*
7.4 Ungerboeck's TCM Design Rules *216*
7.5 Tcm Decoder *220*
7.6 Performance Evaluation for Awgn Channel *221*
7.7 Computation of d_{free} *227*
7.8 Tcm for Fading Channels *228*
7.9 Concluding Remarks *232*
Summary *233*
Problems *234*
Computer Problems *238*

Part III
Coding for Secure Communications

8. Cryptography **241**
8.1 Introduction to Cryptography *241*
8.2 An Overview of Encryption Techniques *242* 密码概念
8.3 Operations Used By Encryption Algorithms *245*
8.4 Symmetric (Secret Key) Cryptography *246*
8.5 Data Encryption Standard (DES) *248*
8.6 International Data Encryption Algorithm (IDEA) *252*
8.7 RC Ciphers *253*
8.8 Asymmetric (Public-Key) Algorithms *254*
8.9 The RSA Algorithm *254*
8.10 Pretty Good Privacy (PGP) *256*
8.11 One-Way Hashing *259*
8.12 Other Techniques *260*
8.13 Secure Communication Using Chaos Functions *261*

8.14 Cryptanalysis *262*
8.15 Politics of Cryptography *264*
8.16 Concluding Remarks *265*
Summary *268*
Problems *269*
Computer Problems *271*

Index **273**

PART I

Information Theory and Source Coding

1

Source Coding

Not everything that can be counted counts, and not everything that counts can be counted.

—Albert Einstein (1879-1955)

Albert Einstein

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 the cellular phones is a common sight, sometimes even in cinema theatres. Movies can be rented in the form of a DVD disk. Email addresses and web addresses are common on business cards. Many people prefer to send emails 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 a key to success, but in today's world it is *the* key). And behind all this information and its exchange lie the tiny 1's and 0's (the omnipresent bits) that hold information by merely the way they sit next to one another. Yet the information age that we live in today owes its existence, primarily, to a seminal paper published in 1948 that laid the foundation of the wonderful field of **Information Theory**–a theory 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 includes the content of any of the standard communication media, such as telegraphy, telephony, radio, or television, and the signals of electronic computers, servo-mechanism systems, and other data-processing devices. The theory is even applicable to the signals of the nerve networks of humans and other animals.

The chief concern of information theory is to discover mathematical laws governing systems designed to communicate or manipulate information. It sets up quantitative measures of information and of the capacity of various systems to transmit, store, and otherwise process information. Some of the problems treated are related to finding the best methods of using various available communication systems and the best methods for separating wanted information or signal, from extraneous information or noise. Another problem is the setting of upper bounds on the capacity of 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 boundaries of information theory are quite fuzzy. The theory overlaps heavily with communication theory but is more oriented towards the fundamental limitations on the processing and communication of information and less towards the detailed operation of the devices employed.

In this chapter, we shall first develop an intuitive understanding of information. It will be followed by 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 two source coding techniques, the **Huffman 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. 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, analog or digital, produces an output that is random in nature. If it were not random, i.e., the output were known exactly, there would be no need to transmit it! 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 email).

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

Coding

(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 this winter.

The three sentences carry different amounts of information. In fact, the first sentence hardly carries any information. Everybody knows that the sun rises in the East and the probability of this happening again is almost unity. 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 have 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 but carries the minimum information.

We will now develop a mathematical measure of information.

Definition 1.1 **Consider a discrete random variable X with possible outcomes x_i, $i = 1, 2, \ldots, n$.**

The Self-Information of the event $X = x_i$ is defined as

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

We note that a high probability event conveys less information than a low probability event. For an event with $P(x) = 1$, $I(x) = 0$. 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 level of information for physical interpretations throughout this chapter.

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** 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., 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 (H) appears and a 0 if a tail (T) appears. For this source, $P(1) = P(0) = 0.5$. The information content of each output from the source is

$$\begin{aligned} I(x_i) &= -\log_2 P(x_i) \\ &= -\log_2 P(0.5) = 1 \text{ bit} \end{aligned} \tag{1.2}$$

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 bits. 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

$$\begin{aligned} I(x_i) &= -\log_2 P(x_i) \\ &= -\log_2 2^{-m} = m \text{ bits} \end{aligned} \tag{1.3}$$

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 (DMS) (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 source 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 the source A and B are independent. The information content of each output from the source C is

$$\begin{aligned} I(C) &= -\log_2 P(x_i) \\ &= -\log_2 P(0.25) = 2 \text{ bits} \end{aligned} \tag{1.4}$$

We have to use two bits to represent the output from this combined binary source.

Thus, the logarithmic measure of 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

$$P(X = x_i \mid Y = y_j) = P(x_i \mid y_j) \tag{1.5}$$

divided by the probability

$$P(X = x_i) = P(x_i) \tag{1.6}$$

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

$$I(x_i; y_j) = \log\left(\frac{P(x_i|y_j)}{P(x_i)}\right) \tag{1.7}$$

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

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

Therefore,

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

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, $P(x_i \mid y_j) = P(x_i)$, it leads to $I(x_i; y_j) = 0$.

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

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

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. It is a channel that transports 1's and 0's from the transmitter (Tx) to the receiver (Rx). It makes an error occasionally, with probability p. A BSC flips a 1 to 0 and *vice-versa* with equal probability. Let X and Y be binary random variables that represent the input and output of this BSC respectively. Let the input symbols be equally likely and the output symbols depend upon the input according to the channel transition probabilities as given below

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

Fig. 1.1 *A Binary Symmetric Channel.*

It simply implies that the probability of a bit getting flipped (i.e. in error) when transmitted over this

BSC is p. From the channel transition probabilities we have

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

Suppose we are at the receiver and we want to determine what was transmitted at the transmitter, on the basis of what was received. The mutual information about the occurrence of the event $X=0$ given that $Y=0$ 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), then,

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

Hence, from 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 get

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

It is clear from the output that we have *no* information about what was transmitted. Thus, it is a useless channel. For such a channel, we may as well toss a fair coin at the receiver in order to determine what was sent!

Suppose we have a channel where $p = 0.1$, then,

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

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:

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

Fig. 1.2 *A Binary Channel with Asymmetric Probabilities.*

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 we are at the receiver and we want to determine what was transmitted at the transmitter, on the basis of what is received. The mutual information about the occurrence of the event $X = 0$ given that $Y = 0$ 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

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

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 $X = x_i$ on the basis of 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 $p = 0.1$,

$$I(x_0; y_1) = I(0; 1) = \log_2 2(p) = \log_2 2(0.1) = -2.322 \text{ bits.}$$

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

$$I(x_0; y_1) = I(0; 1) = \log_2 2(p) = \log_2 2(1) = -1 \text{ bit.}$$

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

$$I(X; Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j)\, I(x_i, y_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_i) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)} \tag{1.13}$$

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 $I(X; Y) \geq 0$, where equality holds 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

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

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 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!

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

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

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.

Fig. 1.4 *The Binary Entropy Function, $H(X) = -p\log_2(p) - (1-p)\log2(1-p)$.*

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

$$H(X|Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j)\log\frac{1}{P(x_i|y_j)} \qquad (1.16)$$

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

$$I(X;\ Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \qquad (1.17)$$

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 $H(X) = H(X|Y)$, and it is possible if and only if X and Y are statistically independent.

(iii) Since $H(X|Y)$ is the conditional self-information about X given Y and $H(X)$ is the average uncertainty (self-information) of X, $I(X;\ Y)$ is the average uncertainty 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.

Example 1.7 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. 1.5.

Fig. 1.5 *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

$$H(X|Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j)\log\frac{1}{p(x_i|y_j)} \qquad (1.18)$$

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

$$P(x_i, y_j) = P(x_i \mid y_j)\, P(y_j) = P(y_j \mid x_i)\, P(x_i) \qquad (1.19)$$

The plot of $H(X|Y)$ versus q is given in Fig. 1.6 with p as the parameter.

Fig. 1.6 *The Plot of Conditional Entropy H(X| Y) Versus q.*

The average mutual information $I(X; Y)$ is given in Fig. 1.7. 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 $p \leq 0.5$), the mutual information between the random variable X (at the transmitter) and the random variable Y (receiver) decreases.

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 1.7 The **average mutual information** between two continuous random variables X and Y is defined as

$$I(X;Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x)p(y|x)\log\frac{p(y|x)p(x)}{p(x)p(y)}\,dxdy \tag{1.20}$$

Fig. 1.7 *The Plot of the Average Mutual Information I(X; Y) Versus q.*

It should be pointed out that the definition of average mutual information can be carried over from discrete random variables to 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 1.8 The **differential entropy** of a continuous random variable X is defined as

$$H(X) = -\int_{-\infty}^{\infty} p(x)\log p(x) \tag{1.21}$$

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 1.9 The **Average Conditional Entropy** of a continuous random variables X given Y is defined as

$$H(X|Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x, y)\log p(x|y)\,dxdy \tag{1.22}$$

The average mutual information can be expressed as

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{1.23}$$

1.5 SOURCE CODING THEOREM

In this section we explore efficient representation (efficient coding) of symbols generated by a source. The primary objective is the compression of data by efficient representation of the symbols. Suppose a discrete memoryless source (DMS) outputs a symbol every t seconds and 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

$$H(X) = \sum_{j=1}^{L} P(x_i)\log_2 P(x_i) \leq \log_2 L \tag{1.24}$$

The equality holds when the symbols are equally likely. It means 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 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 1.10 A **code** is a set of vectors called **codewords**.

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

$$R = \log_2 L, \tag{1.25}$$

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

$$R = \lfloor \log_2 L \rfloor + 1. \tag{1.26}$$

As we saw earlier, to encode the letters of the English alphabet, we need $R = \lfloor \log_2 26 \rfloor + 1 = 5$ bits. The **FLC** for the English alphabet suggests that every letter in the alphabet is equally important (probable) and hence each one requires 5 bits for representation. However, we know that some letters are less common (x, q, z etc.) while 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 is to use a **Variable Length Code (VLC)**.

Example 1.8 Suppose we have only the first eight letters of the English alphabet (A–H) in our vocabulary. The **Fixed Length Code (FLC)** for this set of letters would be

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

Fixed Length Code

A **VLC** for the same set of letters can be

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

Variable Length Code 1

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 number of bits simply because the letters appearing more frequently in the pseudo sentence are represented with fewer number of bits.

We look at yet another **VLC** for the first 8 letters of the English alphabet:

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

Variable Length Code 2

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 here-0 1001 0001 which is used to represent A BAD CAB. 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 [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 one codeword (symbol) ends and the next one begins, since the lengths of the codewords are variable. However, this problem does not exist with VLC1. Here no codeword forms the prefix of any other codeword. This is called the **prefix condition**. 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 called **Uniquely Decodable** or **Instantaneous Codes** cause no decoding delay. In this example, the **VLC2** is *not* a uniquely decodable code, hence not a code of any utility. The **VLC1** is uniquely decodable, though less economical in terms of bits per symbol.

Definition 1.11 A **Prefix Code** is one in which no codeword forms the prefix of any other codeword. Such codes are also called **Uniquely Decodable** or **Instantaneous Codes**.

We now proceed 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

$$\bar{R} = \sum_{k=1}^{L} n(x_k)P(x_k) \tag{1.27}$$

where $n(x_i)$ is the length of the codeword 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

$$\sum_{k=1}^{L} 2^{-n_k} \le 1 \tag{1.28}$$

Proof First we prove the sufficient condition. Consider a binary tree of order (depth) $n = n_L$. This tree has 2^{n_L} terminal nodes as depicted in Fig. 1.8. 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

$$\sum_{k=1}^{j} 2^{-n_k} < \sum_{k=1}^{L} 2^{-n_k} \le 1. \tag{1.29}$$

Thus, we can 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 the figure.

Fig. 1.8 *A Binary Tree of Order n_L.*

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

$$\sum_{k=1}^{L} 2^{-n_k} \le 2^n \tag{1.30}$$

This leads to

$$\sum_{k=1}^{L} 2^{-n_k} \le 1 \tag{1.31}$$

Example 1.9 Consider the construction of a prefix code using a binary tree.

Fig. 1.9 *Constructing a Binary Prefix Code using a Binary Tree.*

We start from the mother node and proceed toward the terminal nodes of the binary tree (Fig. 1.9). Let the mother node be labelled '0' (could have been labelled '1' as well). Each node gives rise to two branches (binary tree). Let's 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 labelled c_1.

Next, we 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

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

Thus, 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 set of letters from a DMS with finite entropy $H(X)$ and x_k, $k = 1, 2, ..., L$ the output symbols, occurring with probabilities $P(x_k)$, $k = 1, 2, ..., L$. Given these parameters, it is possible to construct a code that satisfies the prefix condition and has an average length $\bar{R}$ that satisfies the inequality

$$H(X) \le \bar{R} < H(X) + 1 \tag{1.32}$$

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

$$H(X) - \bar{R} = \sum_{k=1}^{L} p_k \log_2 \frac{1}{p_k} - \sum_{k=1}^{L} p_k n_k = \sum_{k=1}^{L} p_k \log_2 \frac{2^{-n_k}}{p_k}$$

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

$$H(X) - \bar{R} \le (\log_2 e) \sum_{k=1}^{L} p_k \left(\frac{2^{-n_k}}{p_k} - 1 \right)$$

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

The last inequality follows from the Kraft inequality. Equality holds if and only if $p_k = 2^{-n_k}$ for $1 \le k \le L$.

Thus the lower bound is proved. Next, we prove the upper bound. Let us select the codeword lengths n_k such that $2^{-n_k} \le p_k < 2^{-n_k + 1}$. First consider $2^{-n_k} \le p_k$. Summing both sides over $1 \le k \le L$ gives us

$$\sum_{k=1}^{L} 2^{-n_k} \le \sum_{k=1}^{L} p_k = 1$$

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

Next consider $p_k < 2^{-n_k+1}$. Take logarithm on both sides to get

$$\log_2 p_k < -n_k + 1,$$

or,

$$n_k < 1 - \log_2 p_k.$$

On multiplying both sides by p_k and summing over $1 \le k \le L$ we obtain

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

or,

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

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 $\bar{R} = H(X)$ 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 higher entropy (uncertainty) requires, on an average, more number of bits to represent the source symbols in terms of a prefix code.

Definition 1.12 The **efficiency** of a prefix code is defined as

$$\eta = \frac{H(x)}{\bar{R}} \tag{1.33}$$

It is clear from the source coding theorem that the efficiency of a prefix code $\eta \le 1$. Efficient representation of symbols leads to **compression** of data. Source coding is primarily used for compression of data (and images).

Example 1.10 Consider a source X which generates four symbols with probabilities $p_1 = 0.5$, $p_2 = 0.3$, $p_3 = 0.1$ and $p_4 = 0.1$. The entropy of this source is

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

Suppose we use the prefix code {0, 10, 110, 111} constructed in Example 1.9. Then the average codeword length, $\bar{R}$ is given by

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

Thus we have

$$H(X) \le \bar{R} \le H(X) + 1$$

The efficiency of this code is $\eta = (1.685/1.700) = 0.9912$. Had the source symbol probabilities been $p_k = 2^{-n_k}$, i.e., $p_1 = 2^{-1} = 0.5$, $p_2 = 2^{-2} = 0.25$, $p_3 = 2^{-3} = 0.125$ and $p_4 = 2^{-3} = 0.125$, the average codeword length would be, $\bar{R} = 1.750$ 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 $P(x_i)$, $i = 1, 2, ..., L$. The algorithm is optimal in the sense that the average number of bits it requires to represent the source symbols

is a minimum, and also meets the prefix condition. The steps of the Huffman coding algorithm are given below:

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

(ii) Take the bottom two symbols and tie them together as shown below. 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. 1.10.

Fig. 1.10 *Combining Probabilities in Huffman Coding.*

(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 1.11 Consider a DMS with seven possible symbols x_i, $i = 1, 2, ..., 7$ and the corresponding probabilities $p_1 = 0.37$, $p_2 = 0.33$, $p_3 = 0.16$, $p_4 = 0.07$, $p_5 = 0.04$, $p_6 = 0.02$, and $p_7 = 0.01$. We first arrange the probabilities in the decreasing order and then construct the Huffman tree as in Fig. 1.11.

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. 1.11 *Huffman Coding for Example 1.11.*

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

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

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

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

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

Example 1.12 This example shows that Huffman coding is not unique. Consider a DMS with seven possible symbols x_i, $i = 1, 2, ..., 7$ and the corresponding probabilities $p_1 = 0.46$, $p_2 = 0.26$, $p_3 = 0.12$, $p_4 = 0.06$, $p_5 = 0.03$, $p_6 = 0.02$, and $p_7 = 0.01$.

Symbol	Probability	Self Information	Codeword
x_1	0.46	1.1203	1
x_2	0.30	1.7370	00
x_3	0.12	3.0589	010
x_4	0.06	4.0589	0110
x_5	0.03	5.0589	01110
x_6	0.02	5.6439	011110
x_7	0.01	6.6439	011111

Fig. 1.12 *Huffman Coding for Example 1.12.*

The entropy of the source is found out to be

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

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

$$\bar{R} = \sum_{k=1}^{7} n(x_k)P(x_k)$$

$$= 1(0.46) + 2(0.30) + 3(0.12) + 4(0.06) + 5(0.03) + 6(0.02) + 6(0.01)$$

$$= 1.9900 \text{ bits}.$$

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

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 than, 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 yields 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 than, 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. 1.13, 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. 1.13 *Alternative way of Huffman Coding in Example 1.12 which Leads to a Different Code.*

Symbol	Probability	Self Information	Codeword
x_1	0.46	1.1203	1
x_2	0.30	1.7370	00
x_3	0.12	3.0589	011
x_4	0.06	4.0589	0101
x_5	0.03	5.0589	01001
x_6	0.02	5.6439	010000
x_7	0.01	6.6439	010001

The entropy of the source is

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

and the average number of bits per symbol is

$$\bar{R} = \sum_{k=1}^{7} n(x_k)P(x_k)$$

$$= 1(0.46) + 2(0.30) + 3(0.12) + 4(0.06) + 5(0.03) + 6(0.02) + 6(0.01)$$

$$= 1.9900 \text{ bits}.$$

The efficiency of this code is $\eta = (1.9781/1.9900) = 0.9940$. Thus both codes are equally efficient.

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 becomes

$$BH(X) \le \bar{R}_B < BH(X) + 1$$

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

$$H(X) \le \frac{\overline{R_B}}{B} < H(X) + \frac{1}{B} \tag{1.34}$$

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 1.13 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

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

The average number of binary digits per symbol is

$$\begin{aligned}\overline{R} &= \sum_{k=1}^{3} n(x_k)P(x_k) \\ &= 1(0.40) + 2(0.35) + 2(0.25) = 1.60 \text{ bits,}\end{aligned}$$

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 Pairs	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

$$\begin{aligned}2H(X) &= -\sum_{k=1}^{9} p_k \log_2 p_k = 3.1177 \text{ bits,} \\ \Rightarrow H(X) &= 1.5589 \text{ bits.}\end{aligned}$$

Note that the source entropy has not changed ! The average number of bits per block (symbol pair) is

$$\begin{aligned}\overline{R_B} &= \sum_{k=1}^{9} n(x_k)P(x_k) \\ &= 2(0.1600) + 3(0.1400) + 3(0.1400) + 3(0.1225) \\ &\quad + 3(0.1000) + 4(0.1000) + 4(0.0875) + 4(0.0875) + 4(0.0625) \\ &= 3.1775 \text{ bits per symbol pair.} \\ \Rightarrow \overline{R} &= 3.1775/2 = 1.5888 \text{ bits per symbol.}\end{aligned}$$

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 1.14 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

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

The average number of bits per symbol is

$$\begin{aligned}\overline{R} &= \sum_{k=1}^{3} n(x_k)P(x_k) \\ &= 1(0.50) + 2(0.30) + 2(0.20) = 1.50 \text{ bits,}\end{aligned}$$

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 as follows.

Symbol Pairs	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

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

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

The average number of bits per block (symbol pair) is

$$\overline{R_B} = \sum_{k=1}^{9} n(x_k)P(x_k)$$

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

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

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 THE LEMPEL-ZIV ALGORITHM

Huffman coding requires the 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, while Huffman coding is optimal for a DMS source where the occurrence of one symbol does not alter the probabilities of the subsequent symbols, it is not the best choice for a source with

memory. For example, consider the problem of compression of written text. We know that many letters occur in pairs or groups, like 'q-u', 't-h', 'i-n-g' etc. It would 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 (called innovation 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 1.15 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. 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), and so on.

Thus the coded version of the above string is:

0001000000100011010101111010110 1.

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 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.

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

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 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 per cent, whereas the Huffman code compresses the transmission by only 43 per cent.

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

1.8 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 compared to other compression methods, but it is easy to implement and is quick to execute. RLE is supported by most bitmap file formats such as TIFF, BMP and PCX.

Example 1.16 Consider the following bit stream:

111111111111111000000000000000000001111.

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 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 (3×64) 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 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 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 image compression methods and a means to identify which has been applied. We will restrict our attention here to only one of the methods, for 256-colour 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 colour 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 colour 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 colours in a palette are used in an image, then, we need all 256 values of a byte to represent those colours. 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 to leave room for the flag/count combinations, we might double the size of 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 colours. So, they optimized their compression scheme for the case of up to 192 colors only. Images with more colours will also probably get good compression, just not quite as good, with this scheme.

Example 1.17 PCX compression encodes single occurrences of colour (that is, a pixel that is not part of a run of the same colour) 0 through 191 simply as the binary byte representation of exactly that numerical value. Consider 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 colour 192 (and all the colours 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 colour. In fact, if we have a run of pixels of one of the colours with palette code even over 191, we can still code the run easily since the top two bits are not reserved in this second, colour 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 code additional runs of that pixel until we exhaust all pixels in the run. The next question is: how do we code those remaining colours 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 colours 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. Thus, quantization 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 quantized.

1.9 RATE DISTORTION FUNCTION

Although we live in an analog world, most of the communication takes place in digital form. Since most natural sources (e.g. speech, video etc.) are analog, they are first sampled, quantized and then processed. 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 quantized in amplitude and encoded as a sequence of bits. A simple encoding strategy can be to define L levels and encode every sample using

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

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. Quantization of amplitude results in data compression at the cost of signal integrity. It's a form of lossy data compression where some measure of difference between the actual source samples $\{x_k\}$ and the corresponding quantized value $\{\tilde{x}_k\}$ results from distortion.

Definition 1.13 The **squared-error distortion** is defined as

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

In general a distortion measure may be represented as

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

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.,

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

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

Definition 1.14 The **distortion** between a sequence of n samples, X_n, and their corresponding n quantized values, $\tilde{X}_n$ is defined as

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

It has been assumed here that the random process is stationary.

Next, let a memoryless source have a continuous output X and the quantized 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 1.15 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

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

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.

Theorem 1.3 The minimum information rate necessary to represent the output of a discrete time, continuous amplitude memoryless Gaussian source with variance σ_x^2, based on a mean square-error distortion measure per symbol, is

$$R_g(D) = \begin{cases} \frac{1}{2}\log_2(\sigma_x^2 / D) & 0 \le D \le \sigma_x^2 \\ 0 & D > \sigma_x^2 \end{cases}$$

Consider the two cases:

(i) $D \ge \sigma_x^2$: For this case there is no need to transfer any information. For the reconstruction of the samples (with distortion greater than or equal to the variance) one can use statistically independent, zero mean Gaussian noise samples with variance $D = \sigma_x^2$.

(ii) $D < \sigma_x^2$: For this case the number of bits per output symbol decreases monotonically as D increases. The plot of the rate distortion function is given in Fig. 1.14.

Fig. 1.14 *Plot of the $R_g(D)$ versus D/σ_x^2.*

Theorem 1.4 There exists an encoding scheme that maps the source output into codewords such that for any given distortion D, the minimum rate $R(D)$ bits per sample is sufficient to reconstruct the source output with an average distortion that is arbitrarily close to D.

Thus, the distortion function for any source gives the lower bound on the source rate that is possible for a given level of distortion.

Definition 1.16 The **distortion rate function** for a discrete time, memoryless gaussian source is defined as

$$D_g(R) = 2^{-2R}\sigma_x^2.$$

Example 1.18 For a discrete time, memoryless Gaussian source, the distortion (in dB) as a function of its variance can be expressed as

$$10 \log_{10} D_g(R) = -6R + 10 \log_{10} \sigma_x^2.$$

Thus the mean square distortion decreases at a rate of 6 dB/bit.

The rate distortion function of a discrete time, memoryless continuous amplitude source with zero mean and finite variance σ_x^2 with respect to the mean square error distortion measure D is upper bounded as

$$R(D) \le \frac{1}{2}\log_2(D) \quad 0 \le D \le \sigma_x^2.$$

This upper bound can be intuitively understood as follows. We know that for a given variance, the zero mean Gaussian random variable exhibits the maximum differential entropy attainable by any random variable. Hence, for a given distortion, the minimum number of bits per sample required is upper bounded by the gaussian random variable.

The next obvious question is: What would be a good design for a quantizer? Is there a way to construct a quantizer that minimizes the distortion without using too many bits? We shall find the answers to these questions in the next section.

1.10 OPTIMUM QUANTIZER DESIGN

In this section, we look at optimum quantizer design. Consider a continuous amplitude signal whose amplitude is not uniformly distributed, but varies according to a certain probability density function, $p(x)$. We wish to design the optimum scalar quantizer that minimizes some function of the quantization error $q = \tilde{x} - x$, where $\tilde{x}$ is the quantized value of x. The distortion resulting due to the quantization 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 quantizer is one that minimizes D by optimally selecting the output levels and the corresponding input range of each output level. The resulting optimum quantizer is called the **Lloyd-Max Quantizer.** For an L-level quantizer the distortion is given by

$$D = \sum_{k=1}^{L} \int_{x_{k-1}}^{x_k} f(\tilde{x} - x)\, p(x)\, dx$$

The necessary conditions for minimum distortion are obtained by differentiating D with respect to $\{x_k\}$ and $\{\tilde{x}_k\}$. As a result of the differentiation process we end up with the following system of equations

$$f(\tilde{x}_k - x) = f(\tilde{x}_{k+1} - x_k), \qquad k = 1, 2, \cdots, L-1$$

$$\int_{x_{k-1}}^{x_k} f'(\tilde{x}_{k+1} - x_k)\, p(x)\, dx \qquad k = 1, 2, \cdots, L$$

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

$$x_k = \frac{1}{2}(\tilde{x}_k - \tilde{x}_{k+1}), \qquad k = 1, 2, \cdots, L-1$$

$$\int_{x_{k-1}}^{x_k} (\tilde{x}_k - x)\, p(x)\, dx = 0, \qquad k = 1, 2, \cdots, L$$

The nonuniform quantizers are optimized with respect to the distortion. However, each quantized sample is represented by *equal* number of bits (say, R bits/sample). It is possible to have a more efficient **VLC**. The discrete source outputs that result from quantization is characterized by a set of probabilities p_k. These probabilities are then used to design efficient **VLC** (source coding). In order to compare the performance of different nonuniform quantizers, we first fix the distortion, D, and then compare the average number of bits required per sample.

Example 1.19 Consider an eight level quantizer 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 minimization, the values x_k and $\tilde{x}_k$ are listed in Table 1.3.

Table 1.3 *Optimum quantization and Huffman coding*

Level,	x_k	$\tilde{x}_k$	p_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 quantizer 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.

1.11 INTRODUCTION TO IMAGE COMPRESSION

Earlier in this chapter we discussed the coding of data sets for compression. By applying these techniques we can store or transmit all of the information content of a string of data with fewer bits than are in the source data. The minimum number of bits that we must use to convey all the information in the source data is determined by the entropy measure of the source. Good compression ratios can be obtained via entropy encoders and universal encoders for sufficiently large source data blocks. In this section, we look at compression techniques used to store and transmit image data.

Images can be sampled and quantized sufficiently finely so that a binary data stream can represent the original data to an extent that is satisfactory to the most discerning eye. Since we can represent a picture by anything from a thousand to a million bytes of data, we should be able to apply the techniques studied earlier directly to the task of compressing that data for storage and transmission. First, we consider the following points:

1. High quality images are represented by very large data sets. A photographic quality image may require 40 to 100 million bits for representation. These large file sizes drive the need for extremely high compression ratios to make storage and transmission (particularly of movies) practical.
2. Applications that involve imagery such as television, movies, computer graphical user interfaces, and the World Wide Web need to be fast in execution and transmission across

distribution networks, particularly if they involve moving images, to be acceptable to the human eye.

3. Imagery is characterised by higher redundancy than is true of other data. For example, a pair of adjacent horizontal lines in an image is nearly identical while, two adjacent lines in a book are generally different.

The first two points indicate that the highest level of compression technology needs to be used for the movement and storage of image data. The third factor indicates that high compression ratios could be applied. The third factor also says that some special compression techniques may be possible to take advantage of the structure and properties of image data. The close relationship between neighbouring pixels in an image can be exploited to improve the compression ratios. This has a very important implication for the task of coding and decoding image data for real-time applications.

Another interesting point to note is that the human eye is highly tolerant to approximation error in an image. Thus, it may be possible to compress the image data in a manner in which the less important details (to the human eye) can be ignored. That is, by trading off some of the quality of the image we might obtain a significantly reduced data size. This technique is called **Lossy Compression**, as opposed to the **Lossless Compression** techniques discussed earlier. Such liberty cannot be taken, say, financial or textual data! **Lossy Compression** can only be applied to data such as images and audio where deficiencies are made up by the tolerance by human senses of sight and hearing.

1.12 THE JPEG STANDARD FOR LOSSLESS COMPRESSION

The Joint Photographic Experts Group (JPEG) was formed jointly by two 'standards' organisations—the CCITT (The European Telecommunication Standards Organisation) and the International Standards Organisation (ISO). Let us now consider the lossless compression option of the **JPEG Image Compression Standard** which is a description of 29 distinct coding systems for compression of images. Why are there so many approaches? It is because the needs of different users vary so much with respect to quality versus compression and compression versus computation time that the committee decided to provide a broad selection from which to choose. We shall briefly discuss here two methods that use entropy coding.

The two lossless JPEG compression options discussed here differ only in the form of the entropy code that is applied to the data. The user can choose either a **Huffman Code** or an **Arithmetic Code**. We will not treat the Arithmetic Code concept in much detail here. However, we will summarize its main features:

Arithmetic Code, like Huffman Code, achieves compression in transmission or storage by using the probabilistic nature of the data to render the information with fewer bits than used in the source data stream. Its primary advantage over the Huffman Code is that it comes closer to the Shannon entropy limit of compression for data streams that involve a relatively small alphabet. The reason is that Huffman codes work best (highest compression ratios) when the

probabilities of the symbols can be expressed as fractions of powers of two. The Arithmetic code construction is not closely tied to these particular values, as is the Huffman code. The computation of coding and decoding Arithmetic codes is costlier than that of Huffman codes. Typically a 5 to 10% reduction in file size is seen with the application of Arithmetic codes over that obtained with Huffman coding.

Some compression can be achieved if we can predict the next pixel using the previous pixels. In this way we just have to transmit the prediction coefficients (or difference in the values) instead of the entire pixel. The predictive process that is used in the lossless JPEG coding schemes to form the innovations data is also variable. However, in this case, the variation is not based upon the user's choice, but rather, for any image on a line by line basis. The choice is made according to that prediction method that yields the best prediction overall for the entire line.

There are eight prediction methods available in the JPEG coding standards. One of the eight (which is the no prediction option) is not used for the lossless coding option that we are examining here. The other seven may be divided into the following categories:

1. Predict the next pixel on the line as having the same value as the last one.
2. Predict the next pixel on the line as having the same value as the pixel in this position on the previous line (that is, above it).
3. Predict the next pixel on the line as having a value related to a combination of the previous, above and previous to the above pixel values. One such combination is simply the average of the other three.

The differential encoding used in the JPEG standard consists of the differences between the actual image pixel values and the predicted values. As a result of the smoothness and redundancy present in most pictures, these differences give rise to relatively small positive and negative numbers that represent the small typical error in the prediction. Hence, the probabilities associated with these values are large for the small innovation values and quite small for large ones. This is exactly the kind of data stream that compresses well with an entropy code.

The typical lossless compression for natural images is 2:1. While this is substantial, it does not in general solve the problem of storing or moving large sequences of images as encountered in high quality video.

1.13 THE JPEG STANDARD FOR LOSSY COMPRESSION

The JPEG standard includes a set of sophisticated lossy compression options developed after a study of image distortion acceptable to human senses. The JPEG lossy compression algorithm consists of an image simplification stage, which removes the image complexity at some loss of fidelity, followed by a lossless compression step based on predictive filtering and Huffman or Arithmetic coding.

The lossy image simplification step, which we will call the image reduction, is based on the exploitation of an operation known as the Discrete Cosine Transform (DCT), defined as follows.

$$Y(k, l) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} 4y(i, j) \cos\left(\frac{\pi k}{2N}(2i+1)\right) \cos\left(\frac{\pi l}{2M}(2j+1)\right)$$

where the input image is N pixels by M pixels, $y(i, j)$ is the intensity of the pixel in row i and column j, $Y(k,l)$ is the DCT coefficient in row k and column l of the DCT matrix. All DCT multiplications are real. This lowers the number of required multiplications, as compared to the Discrete Fourier Transform. For most images, much of the signal energy lies at low frequencies, which appear in the upper left corner of the DCT. The lower right values represent higher frequencies, and are often small (usually small enough to be neglected with little visible distortion).

In the JPEG image reduction process, the DCT is applied to 8 by 8 pixel blocks of the image. Hence, if the image is 256 by 256 pixels in size, we break it into 32 by 32 square blocks of 8 by 8 pixels and treat each one independently. The 64 pixel values in each block are transformed by the DCT into a new set of 64 values. These new 64 values, known also as the DCT coefficients, form a whole new way of representing an image. The DCT coefficients represent the spatial frequency of the image sub-block. The upper left corner of the DCT matrix has low frequency components and the lower right corner the high frequency components (see Fig. 1.15). The top left coefficient is called the DC coefficient. Its value is proportional to the average value of the 8 by 8 block of pixels. The rest are called the AC coefficients.

So far we have not obtained any reduction simply by taking the DCT. However, due to the nature of most natural images, maximum energy (information) lies in low frequency as opposed to high frequency. We can represent the high frequency components coarsely, or drop them altogether, without strongly affecting the quality of the resulting image reconstruction. This leads to a lot of compression (lossy). The JPEG lossy compression algorithm does the following operations:

1. First the lowest weights are trimmed by setting them to zero.
2. The remaining weights are quantized (that is, rounded off to the nearest of some number of discrete code represented values), some more coarsely than others according to observed levels of sensitivity of viewers to these degradations.

Fig. 1.15 *Typical Discrete Cosine Transform (DCT) Values.*

Now several lossless compression steps are applied to the weight data that results from the above DCT and quantization process, for all the image blocks. We observe that the DC coefficient, which represents the average image intensity, tends to vary slowly from one block of 8×8 pixels to the next. Hence, the prediction of this value from surrounding blocks works well. We just need to send one DC coefficient and the difference between the DC coefficients of successive blocks. These differences can also be source coded.

We next look at the AC coefficients. We first quantize them, which transforms most of the high frequency coefficients to zero. We then use a **zig-zag** coding as shown in Fig. 1.16. The purpose of the zig-zag coding is that we gradually move from the low frequency to high frequency, avoiding abrupt jumps in the values. Zig-zag coding will lead to long runs of 0's, which are ideal for RLE followed by Huffman or Arithmetic coding.

Fig. 1.16 *An Example of Quantization followed by Zig-zag Coding.*

The typically quoted performance for JPEG is that photographic quality images of natural scenes can be preserved with compression ratios of up to about 20:1 or 25:1. Usable quality (that is, for noncritical purposes) can result for compression ratios in the range of 200:1 up to 230:1.

1.14 CONCLUDING REMARKS

In 1948, Shannon published his landmark paper titled "A Mathematical Theory of Communication". He begins this pioneering paper on information theory by observing that *the fundamental problem of* communication is that of reproducing at one point either exactly or approximately a message *selected at another point.* He then proceeds so thoroughly to establish the foundations of information theory that his framework and terminology remain standard. Shannon's theory was an immediate success with communications engineers and stimulated the growth of a technology which led to today's Information Age. Shannon published many more provocative and influential articles in a variety of disciplines. His master's thesis, "A Symbolic Analysis of Relay and Switching Circuits", used Boolean algebra to establish the theoretical underpinnings of digital circuits. This work has broad significance because digital circuits are fundamental to the operation of modern computers and telecommunications systems.

Shannon was renowned for his eclectic interests and capabilities. A favourite story describes him juggling while riding a unicycle down the halls of Bell Labs. He designed and built chess-playing, maze-solving, juggling and mind-reading machines. These activities bear out Shannon's claim that he was more motivated by curiosity than usefulness. In his words "I just wondered how things were put together."

The Huffman code was created by American Scientist, D. A. Huffman in 1952. Modified Huffman coding is today used in the Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MEPG) standards.

A very efficient technique for encoding sources without needing to know their probable occurrence was developed in the 1970s by the Israelis Abraham Lempel and Jacob Ziv. The compress and uncompress utilities of the UNIX operating system use a modified version of this algorithm. The GIF format (Graphics Interchange Format), developed by CompuServe, involves simply an application of the Lempel-Ziv-Welch (LZW) universal coding algorithm to the image data.

And finally, to conclude this chapter we mention that Shannon, the father of Information Theory, passed away on February 24, 2001. Excerpts from the obituary published in the New York Times:

Claude Shannon, Mathematician, Dies at 84

By GEORGE JOHNSON

Dr. Claude Elwood Shannon, the American mathematician and computer scientist whose theories laid the groundwork for the electronic communications networks that now lace the earth, died on Saturday in Medford, Mass...

SUMMARY

- The Self-Information of the event $X = x_i$ is given by $I(x_i) = \log\left(\frac{1}{P(x_i)}\right) = -\log P(x_i)$.
- The Mutual Information $I(x_i; y_j)$ between x_i and y_j is given by $I(x_i; y_j) = \log\left(\frac{P(x_i|y_j)}{P(x_i)}\right)$.
- The Conditional Self-Information of the event $X = x_i$ given $Y = y_j$ is defined as $I(x_i \mid y_j) = \log\left(\frac{1}{P(x_i|y_j)}\right) = -\log P(x_i \mid y_j)$.
- The Average Mutual Information between two random variables X and Y is given by $I(X; Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j) I(x_i; y_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)}$. For the case when X and Y are statistically independent, $I(X; Y) = 0$. The average mutual information $I(X; Y) \geq 0$, with equality if and only if X and Y are statistically independent.

- The **Average Self-Information** of a random variable X is given by $H(X) = \sum_{i=1}^{n} P(x_i) I(x_i) = -\sum_{i=1}^{n} P(x_i) \log P(x_i)$. $H(X)$ is called the entropy.
- The **Average Conditional Self-Information** called the **Conditional Entropy** is given by

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

- $I(x_i; y_j) = I(x_i) - I(x_i \mid y_j)$ and $I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. Since $I(X; Y) \geq 0$, it implies that $H(X) \geq H(X|Y)$.
- The **Average Mutual Information** between two continuous random variables X and Y is given by $I(X; Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x)p(y|x) \log \frac{p(y|x)p(x)}{p(x)p(y)}\, dx\, dy$
- The **Differential Entropy** of a continuous random variables X is given by $H(X) = -\int_{-\infty}^{\infty} p(x) \log p(x)$.
- The **Average Conditional Entropy** of a continuous random variables X given Y is given by $H(X \mid Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x, y) \log p(x|y)\, dx\, dy$.
- A necessary and sufficient condition for the existence of a binary code with codewords having lengths $n_1 \leq n_2 \leq \ldots n_L$ that satisfy the prefix condition is $\sum_{k=1}^{L} 2^{-n_k} \leq 1$. The efficiency of a prefix code is given by $\eta = \frac{H(x)}{\bar{R}}$.
- Let X be the ensemble of letters from a DMS with finite entropy $H(X)$. The **Source Coding Theorem** suggests that it is possible to construct a code that satisfies the prefix condition, and has an average length $\bar{R}$ that satisfies the inequality $H(X) \leq \bar{R} < H(X) + 1$. Efficient representation of symbols leads to compression of data.
- **Huffman Encoding** and **Lempel-Ziv Encoding** are two popular source coding techniques. In contrast to the Huffman coding scheme, the Lempel-Ziv technique is independent of the source statistics. The Lempel-Ziv technique generates a Fixed Length Code, where as the Huffman code is a Variable Length Code.
- **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 most bitmap file formats such as TIFF, BMP and PCX.

- **Distortion** implies some measure of difference between the actual source samples $\{x_k\}$ and the corresponding quantized value $\{\tilde{x}_k\}$. The **squared-error distortion** is given by $d(x_k, \tilde{x}_k) = (x_k - \tilde{x}_k)^2$. In general, a distortion measure may be represented as $d(x_k, \tilde{x}_k) = |x_k - \tilde{x}_k|^p$.
- 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 $R(D) = \min_{p(\tilde{x}|x):E[d(X,\tilde{X})]} I(X, \tilde{X})$ where $I(X, \tilde{X})$; is the average mutual information between X and $\tilde{X}$.
- The distortion resulting due to the **quantization** 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 quantizer is one that minimizes D by optimally selecting the output levels and the corresponding input range of each output level. The resulting **optimum quantizer** is called the **Lloyd-Max quantizer**.
- Quantization 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 Consider an integer valued random variable, X, given by $P(X = n) = \dfrac{1}{An \log^2 n}$, where $A = \sum_{n=2}^{\infty} \dfrac{1}{n \log^2 n}$ and $n = 2, 3 \ldots, \infty$. Find the entropy, $H(X)$.

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

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

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

1.9 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 η of the code?

1.10 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.

1.11 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 η.
(ii) Determine the Huffman code for this source taking *two* symbols at a time and find the efficiency η.
(iii) Determine the Huffman code for this source taking *three* symbols at a time and find the efficiency η.

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

1.13 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 $Z = X + Y$.
(a) Show that $H(Z|X) = H(Y|X)$
(b) If X and Y are independent, then argue that $H(Y) \le H(Z)$ and $H(X) \le H(Z)$. Comment on this observation.
(c) Under what condition will $H(Z) = H(X) + H(Y)$?

1.14 Determine the Lempel Ziv code for the following bit stream
0100111110010100000101010110011000 0.
Recover the original sequence from the encoded stream.

1.15 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

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

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

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

COMPUTER PROBLEMS

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

1.18 Modify the above program so that it can group together n source symbols and then generate the Huffman code. Plot the coding efficiency η 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}.

1.19 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.

1.20 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:

1100000000111100000111111111111111111111100000110000000.

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.

1.21 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 into 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

Channel Capacity and Coding

Experimentalists think that it is a mathematical theorem while the mathematicians believe it to be an experimental fact.

[On the Gaussian curve, remarked to Poincaré:]
Lippman, Gabriel (1845-1921)

2.1 INTRODUCTION

In the previous chapter we saw that most natural sources have inherent redundancies and it is possible to compress data by removing these redundancies using different source coding techniques. After efficient representation of source symbols by the minimum possible number of bits, we transmit these bit-streams over channels (e.g., telephone lines, optical fibres etc.). These bits may be transmitted as they are (for baseband communications), or after modulation (for passband communications). Unfortunately, all real-life channels are noisy. The term noise designates unwanted waves that disturb the transmission and processing of the wanted signals in communication systems. The source of noise may be external to the system (e.g., atmospheric noise, man generated noise etc.), or internal (e.g., thermal noise, shot noise etc.). In effect, the bit stream obtained at the receiver is likely to be different from what was transmitted. In passband communication, the demodulator processes the channel-corrupted waveform and reduces each waveform to a scalar or a vector that represents an estimate of the transmitted data symbols. The detector, which follows the demodulator, decides whether the transmitted bit is a

0 or a 1. This is called **Hard Decision Decoding**. This decision process at the decoder is similar to a binary quantization with two levels. If there are more than 2 levels of quantization, the detector is said to perform a **Soft Decision Decoding**.

The use of hard decision decoding causes an irreversible loss of information at the receiver. Suppose the modulator sends only binary symbols but the demodulator has an alphabet with Q symbols, and assuming the use of quantizer as depicted in Fig. 2.1(a), we have $Q = 8$. Such a channel is called a binary input Q-ary output **Discrete Memoryless Channel**. The corresponding channel is shown in Fig. 2.1(b). The decoder performance depends on the location of the representation levels of the quantizers, which in turn depends on the signal level and the noise power. Accordingly, the demodulator must incorporate automatic gain control in order to realize an effective multilevel quantizer. It is clear that the construction of such a decoder is more complicated than the hard decision decoder. However, soft decision decoding can provide significant improvement in performance over hard decision decoding.

Fig. 2.1 *(a) Transfer Characteristic of Multilevel Quantizer*
(b) Channel Transition Probability Diagram.

There are three balls that a digital communication engineer must juggle: (i) the transmitted signal power, (ii) the channel bandwidth, and (iii) the reliability of the communication system (in terms of the bit error rate). Channel coding allows us to trade-off one of these commodities (signal power, bandwidth or reliability with respect to others). In this chapter, we will study how to achieve reliable communication in the presence of noise. We shall ask ourselves questions like: how many bits per second can be sent over a channel of a given bandwidth and for a given signal to noise ratio (SNR)? For that, we begin by studying a few channel models first.

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.*

The composite **Discrete-input, Discrete-output Channel** is characterized 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. Assuming the noise in the channel causes independent errors in the transmitted binary sequence with average probability of error p

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

A BSC is shown in Fig. 2.3.

Fig. 2.3 *A Binary Symmetric Channel (BSC).*

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 receiver 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 outputs can then be related by a set of qQ conditional probabilities

$$P(Y = y_i \mid X = x_j) = P(y_i \mid x_j), \tag{2.2}$$

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 characterize a DMC can be arranged in the matrix form $\boldsymbol{P} = [p_{ji}]$. $\boldsymbol{P}$ is called the **Probability Transition Matrix** for the channel.

Fig. 2.4 *A Discrete Memoryless Channel (DMC) with q-ary input and Q-ary output.*

In the next section, we will try to answer the question: How many bits can be sent across a given noisy channel, each time the channel is used?

2.3 CHANNEL CAPACITY

Consider a DMC having an input alphabet $X = \{x_0, x_1, ..., x_{q-1}\}$ and an output alphabet $Y = \{x_0, x_1, ..., x_{r-1}\}$. 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 1.2)

$$I(X;Y) = \sum_{j=0}^{q-1} \sum_{i=0}^{r-1} P(x_j)P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)} \tag{2.3}$$

The channel transition probabilities $P(y_i|x_j)$ are determined by the channel characteristics (particularly 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, $I(X;Y)$, maximized over the set of input symbol probabilities $P(x_j)$ is a quantity that depends only on the channel transition probabilities $P(y_i|x_j)$. This quantity is called the **Capacity of the Channel**.

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

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

$$= \max_{P(x_j)} \sum_{j=0}^{q-1} \sum_{i=0}^{r-1} P(x_j)P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)} \tag{2.4}$$

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, $\max C = \max_{P(x_j)} I(X; Y)$, is achieved for $p = 0.5$. From equation (2.4) 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 gets 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 signal to noise ratio (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 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 send 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 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 resort to **Channel formatting.**

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 inverse mapped on 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 in order 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**, and we will use these terms interchangeably. It is interesting to note here that the source coder reduces redundancy to improve efficiency, whereas, 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

$$r = \frac{k}{n} \tag{2.5}$$

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 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 it was first provided by Shannon in his second theorem on 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 DMS has the source alphabet X and entropy $H(X)$ bits per source symbol and the source generates 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 Channel Coding Theorem.

Theorem 2.1 **Channel Coding Theorem (Noisy coding theorem)**

(i) 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

$$\frac{H(X)}{T_s} \le \frac{C}{T_c} \tag{2.6}$$

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

$$\frac{H(X)}{T_s} > \frac{C}{T_c}, \tag{2.7}$$

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 communication 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 the 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 reliable communication (the probability of error as small as desired). From the channel coding theorem, if

$$\frac{1}{T_s} \le \frac{C}{T_c} \tag{2.8}$$

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 code rate of the coder can be expressed as

$$r = \frac{T_c}{T_s} \tag{2.9}$$

Hence, the condition for reliable communication can be rewritten as

$$r \le C \tag{2.10}$$

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 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 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

$$r = \frac{1}{n} \tag{2.11}$$

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 otherwise known as **Majority Decoding**. This also answers the question why n should 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 if a 0 is encoded and sent, there would 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

$$P_e = \sum_{i=m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{2.12}$$

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, Pe
1	10^{-2}
1/3	3×10^{-4}
1/5	10^{-6}
1/7	4×10^{-7}
1/9	10^{-8}
1/11	5×10^{-0}

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 **Additive White Gaussian Noise** (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,

$$Y_k = X_k + N_k, \; k = 1, 2,..., K \tag{2.13}$$

where N_k is the noise sample with zero mean and variance $\sigma^2 = N_0 W$. 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:

$$E[X_k^2] = P, \; k = 1, 2,..., K \tag{2.14}$$

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 maximization has to be done over all distributions on the input X_k that satisfy the power constraint of equation (2.14). Thus, the information capacity of the channel (same as the channel capacity) is given by

$$C = \max_{f_{x_k}(x)} \{I(X; Y) \mid E[X_k^2] = P\}, \tag{2.15}$$

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

Now, from the previous chapter we have,

$$I(X_k; Y_k) = h(Y_k) - h(Y_k \mid X_k) \tag{2.16}$$

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,

$$h(Y_k \mid X_k) = h(N_k) \tag{2.17}$$

Hence we can write Eq. (2.16) as

$$I(X_k; Y_k) = h(Y_k) - h(N_k) \tag{2.18}$$

Since $h(N_k)$ is independent of X_k, maximizing $I(X_k; Y_k)$ translates to maximizing $h(Y_k)$. 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 maximize the mutual information between the channel input X_k and the channel output Y_k, the transmitted signal should also be Gaussian. Therefore we can rewrite (2.15) as

$$C = I(X;Y) \mid E[X_k^2] \; P \text{ and } X_k \text{ is Gaussian} \tag{2.19}$$

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 $P + N_0 W$. It can be shown that the differential entropy of a Gaussian random variable with variance σ^2 is $\frac{1}{2}\log_2(2\pi e\sigma^2)$ (see Problem 2.10). Therefore,

$$h(Y_k) = \frac{1}{2}\log_2[2\pi e(P + N_0 W)] \tag{2.20}$$

and

$$h(N_k) = \frac{1}{2}\log_2[2\pi e(N_0 W)] \tag{2.21}$$

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

$$C = \frac{1}{2}\log_2\left(1 + \frac{P}{N_0 W}\right) \text{ bits per channel use} \tag{2.22}$$

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

$$C = W\log_2\left(1 + \frac{P}{N_0 W}\right) \text{ bits per second} \tag{2.23}$$

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 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, disturbed by Additive White Gaussian Noise of power spectral density $N_0/2$ and limited in bandwidth to W, is given by

$$C = W\log_2\left(1 + \frac{P}{N_0 W}\right) \text{ bits per second}$$

where P is the average transmitted power. This theorem is also called the **Channel Capacity Theorem**.

The **Information Capacity Theorem** is one of the important results in information theory. In a single formula one can see the trade off between the channel bandwidth, the average transmitted power and the noise power spectral density. Given the channel bandwidth and the SNR the channel capacity (bits/second) can be computed. This channel capacity is the fundamental limit on the rate of reliable communication for a power-limited, band-limited Gaussian channel. It should be kept in mind that in order to approach this limit, the transmitted signal must have statistical properties that are Gaussian in nature. Note that the terms channel capacity and information capacity have been used interchangeably.

Let us now derive the same result in a more intuitive manner. Suppose we have a coding scheme that results in an acceptably low probability of error. Let this coding scheme take k information bits and encode them into n bit long codewords. The total number of codewords is $M = 2^k$. Let the average power per bit be P. Thus the average power required to transmit an entire codeword is nP. Let these codewords be transmitted over a Gaussian channel with the noise variance equal to σ^2. The received vector of n bits is also Gaussian with the mean equal to the transmitted codeword and the variance equal to $n\sigma^2$. Since the code is a good one (acceptable error rate), the vector lies inside a sphere of radius $\sqrt{n\sigma^2}$ centred on the transmitted codeword. This sphere itself is contained in a larger sphere of radius $\sqrt{n(P+\sigma^2)}$ where $n(P+\sigma^2)$ is the average power of the received vector.

This concept may be visualized as depicted in Fig. 2.6. There is a large sphere of radius $\sqrt{n(P+\sigma^2)}$ which contains M smaller spheres of radius $\sqrt{n\sigma^2}$. Here $M = 2^k$ is the total number of codewords. Each of these small spheres is centred on a codeword. These are called the **Decoding Spheres**. Any received word lying within a sphere is decoded as the codeword on which the sphere is centred. Suppose a codeword is transmitted over a noisy channel. Then there is a *high* probability that received vector will lie inside the correct decoding sphere (since it is a reasonably good code). The question arises: How many non-intersecting spheres can be packed inside the large sphere? The more number of spheres one can pack, the more efficient will be the code in terms of the code rate. This is known as the **Sphere Packing Problem**.

Fig. 2.6 *Visualization of the Sphere Packing Problem.*

The volume of an n-dimensional sphere of radius r can be expressed as

$$V = A_n r^n \tag{2.24}$$

where A_n is a scaling factor. Therefore, the volume of the large sphere (sphere of all possible received vectors) can be written as

$$V_{all} = A_n\,[n(P+\sigma^2)]^{n/2} \tag{2.25}$$

and the volume of the decoding sphere can be written as

$$V_{ds} = A_n\,[n\sigma^2]^{n/2} \tag{2.26}$$

The maximum number of non intersecting decoding spheres that can be packed inside the large sphere of all possible received vectors is

$$M = \frac{A_n\,[n(P+\sigma^2)]^{n/2}}{A_n\,[n\sigma^2]^{n/2}} = \left(1+\frac{P}{\sigma^2}\right) = 2^{(n/2)\log_2(1+P/\sigma^2)} \tag{2.27}$$

On taking logarithm (base 2) on both sides of the equation we get

$$\log_2 M = \frac{n}{2}\log_2\left(1+\frac{P}{\sigma^2}\right) \tag{2.28}$$

Observing that $k = \log_2 M$, we have

$$\frac{k}{n} = \frac{1}{2}\log_2\left(1+\frac{P}{\sigma^2}\right) \tag{2.29}$$

Note that each time we use the channel, we effectively transmit $\frac{k}{n}$ bits. Thus, the maximum number of bits that can be transmitted *per channel use*, with a low probability of error, is $\frac{1}{2}\log_2 \times \left(1+\frac{P}{\sigma^2}\right)$ as seen previously in Eq. (2.22). Note that σ^2 represents the noise power and is equal to $N_0 W$ for an AWGN with power spectral density $\frac{N_0}{2}$ and limited in bandwidth to W.

2.6 THE SHANNON LIMIT

Consider a Gaussian channel that is limited both in power and bandwidth. We wish to explore the limits of a communication system under these constraints. Let us define an ideal system which can transmit data at a bit rate R_b which is equal to the capacity, C, of the channel, i.e., $R_b = C$. Suppose the energy per bit is E_b. Then the average transmitted power is

$$P = E_b R_b = E_b C \tag{2.30}$$

Therefore, the channel capacity theorem for this ideal system can be written as

$$\frac{C}{W} = \log_2\left(1+\frac{E_b}{N_0}\frac{C}{W}\right) \tag{2.31}$$

This equation can be re-written in the following form

$$\frac{E_b}{N_0} = \frac{2^{C/W} - 1}{C/W} \tag{2.32}$$

The plot of the bandwidth efficiency $\frac{R_b}{W}$ versus $\frac{E_b}{N_0}$ is called the **Bandwidth Efficiency Diagram**, and is given in Fig. 2.7. The ideal system is represented by the line $R_b = C$.

Fig. 2.7 *The Bandwidth Efficiency Diagram.*

The following conclusions can be drawn from the Bandwidth Efficiency Diagram.

(i) For infinite bandwidth, the ratio $\frac{E_b}{N_0}$ tends to the limiting value

$$\left.\frac{E_b}{N_0}\right|_{W\to\infty} = \ln 2 = 0.693 = -1.6 \text{ dB} \tag{2.33}$$

This value is called the **Shannon Limit**. It is interesting to note that the Shannon limit is a fraction. This implies that for very large bandwidths, reliable communication is possible even for the case when the signal power is *less than* the noise power! The channel capacity corresponding to this limiting value is

$$C|_{W\to\infty} = \frac{P}{N_0}\log_2 e \tag{2.34}$$

Thus, at infinite bandwidth, the capacity of the channel is determined by the SNR.

(ii) The curve for the critical rate $R_b = C$ is known as the **Capacity Boundary**. For the case $R_b > C$, reliable communication is not guaranteed. However, for $R_b < C$, there exists some coding scheme which can provide an arbitrarily low probability of error.

(iii) The Bandwidth Efficiency Diagram shows the trade-offs between the quantities $\frac{R_b}{W}, \frac{E_b}{N_0}$ and the probability of error, P_e. Note that for designing any communication system the basic design parameters are the bandwidth available, the SNR and the bit error rate (BER). The BER is determined by the application and the quality of service (QoS) desired. The bandwidth and the power can be traded one for the other to provide the desired BER.

(iv) Any point on the Bandwidth Efficiency Diagram corresponds to an operating point corresponding to a set of values of SNR, Bandwidth Efficiency and BER.

The information capacity theorem predicts the maximum amount of information that can be transmitted through a given bandwidth for a given SNR. We see from Fig. 2.7 that acceptable capacity can be achieved even for low SNRs, provided adequate bandwidth is available. The optimum usage of a given bandwidth is obtained when the signals are noise-like and a minimal SNR is maintained at the receiver. This principle lies in the heart of any spread spectrum communication system, like Code Division Multiple Access (CDMA).

2.7 RANDOM SELECTION OF CODES

Consider a set of M coded signal waveforms constructed from a set of n-dimensional binary codewords. Let us represent these codewords as follows

$$C_i = [c_{i1}\ c_{i2} \ldots c_{in}],\ i = 1, 2, \ldots M \tag{2.35}$$

Since we are considering binary codes, c_{ij} is either a 0 or a 1. Let each bit of the codeword be mapped on to a BPSK waveform p_j so that the codeword may be represented as

$$s_i(t) = \sum_{j=1}^{n} s_{ij}\, p_j(t),\ i = 1,2, \ldots M, \tag{2.36}$$

where

$$s_{ij} = \begin{cases} \sqrt{E} & \text{for } c_{ij} = 1 \\ -\sqrt{E} & \text{for } c_{ij} = 0 \end{cases} \tag{2.37}$$

and $\sqrt{E}$ is the energy per code bit. The waveform $s_i(t)$ can then be represented as the n-dimensional vector

$$s_i = [s_{i1}\ s_{i2} \ldots s_{in}],\ i = 1, 2, \ldots M \tag{2.38}$$

We observe that this corresponds to a hypercube in the n-dimensional space. Let us now encode k bits of information into an n bit long codeword, and map this codeword to one of the M

waveforms. Note that there are a total of 2^k possible waveforms corresponding to the $M = 2^k$ different codewords.

Let the information rate into the encoder be R bits/sec. The encoder takes in k bits at a time and maps the k-bit block to one of the M waveforms. Thus, $k = RT$ and $M = 2^k$ signals are required.

Let us define a parameter D as follows

$$D = \frac{n}{T} \text{ dimensions/sec} \tag{2.39}$$

$n = DT$ is the **dimensionality** of the space. The hypercube mentioned above has $2^n = 2^{DT}$ vertices. Of these, we must choose $M = 2^{RT}$ to transmit the information. Under the constraint $D > R$, the fraction of vertices that can be used as signal points is

$$F = \frac{2^k}{2^n} = \frac{2^{RT}}{2^{DT}} = 2^{-(D-R)T} \tag{2.40}$$

For $D > R$, $F \to 0$ as $T \to \infty$. Since, $n = DT$, it implies that $F \to 0$ as $n \to \infty$. Designing a good coding scheme translates to choosing M vertices out of the 2^n vertices of the hypercube in such a manner that the probability of error tends to zero as we increase n. We saw that the fraction F tends to zero as we choose larger and larger n. This implies that it is possible to increase the minimum distance between these M signal points as $n \to \infty$. Increasing the minimum distance between the signal points would give us the probability of error, $P_e \to 0$.

There are $(2^n)^M$ *distinct* ways of choosing M out of the total 2^n vertices. Each of these choices corresponds to a coding scheme. For each set of M waveforms, it is possible to design a communication system consisting of a modulator and a demodulator. Thus, there are 2^{nM} communication systems, one for each choice of the M coded waveforms. Each of these communication systems is characterized by its probability of error. Of course, many of these communication systems will perform poorly in terms of the probability of error.

Let us pick one of the codes *at random* from the possible 2^{nM} sets of codes. The random selection of this m^{th} codeword occurs with the probability

$$P(\{s_i\}_m) = \frac{1}{2^{nM}} \tag{2.41}$$

Let the corresponding probability of error for this choice of code be $P_e(\{s_i\}_m)$. Then the average probability of error over the ensemble of codes is

$$\bar{P}_e = \sum_{m=1}^{2^{nM}} P_e(\{s_i\}_m) P(\{s_i\}_m).$$

$$= \frac{1}{2^{nM}} \sum_{m=1}^{2^{nM}} P_e(\{s_i\}_m) \tag{2.42}$$

We will next try to upper bound this average probability of error. If we have an upper bound on $\bar{P}_e$, then we can conclude that there exists at least one code for which this upper bound will also hold. Furthermore, if $\bar{P}_e \to 0$ as $n \to \infty$, we can surmise that $P_e(\{s_i\}_m) \to 0$ as $n \to \infty$.

Consider the transmission of a k-bit message $\boldsymbol{X}_k = [x_1\, x_2 \ldots x_k]$ where x_j is binary for $j = 1, 2, \ldots, k$. The conditional probability of error averaged over all possible codes is

$$\overline{P_e(\boldsymbol{X}_k)} = \sum_{\text{all codes}} P_e(\boldsymbol{X}_k, \{s_i\}_m) P(\{s_i\}_m) \tag{2.43}$$

where $P_e(\boldsymbol{X}_k, \{s_i\}_m)$ is the conditional probability of error for a given k-bit message $\boldsymbol{X}_k = [x_1 x_2 \cdots x_k]$, which is transmitted using the code $\{s_i\}_m$. For the mth code,

$$P_e(\boldsymbol{X}_k, \{s_i\}_m) \le \sum_{\substack{l=1 \\ l \ne k}}^{M} P_{2m}(s_l, s_k), \tag{2.44}$$

where, $P_{2m}(s_l, s_k)$ is the probability of error for the binary communication system using signal vectors s_l and s_k. to transmit one of two equally likely k-bit messages. Hence,

$$\overline{P_e(\boldsymbol{X}_k)} \le \sum_{\text{all codes}} P_e(\{s_i\}_m) \sum_{\substack{l=1 \\ l \ne k}}^{M} P_{2m}(s_l, s_k) \tag{2.45}$$

On changing the order of summation we obtain

$$\overline{P_e(\boldsymbol{X}_k)} \le \sum_{\substack{l=1 \\ l \ne k}}^{M} \left[\sum_{\text{all codes}} P_e(\{s_i\}_m) P_{2m}(s_l, s_k) \right] \le \sum_{\substack{l=1 \\ l \ne k}}^{M} \overline{P_2(s_l, s_k)}. \tag{2.46}$$

where $\overline{P_2(s_l, s_k)}$, represents the ensemble average of), $P_{2m}(s_l, s_k)$ over the 2^{nM} codes. For **additive White Gaussian Noise Channel,**

$$P_{2m}(s_l, s_k) = Q\left(\frac{d_{lk}^2}{2N_0}\right) \tag{2.47}$$

where $d_{lk}^2 = |s_l - s_k|^2$ and $Q(x) = \frac{1}{2\pi} \int_x^\infty e^{-x^2/2}\, dx$. If s_l and s_k differ in d coordinates,

$$d_{lk}^2 = |s_l - s_k|^2 = \sum_{j=1}^{n} (s_{lj} - s_{kj})^2 = d(2\sqrt{E})^2 = 4dE \tag{2.48}$$

Therefore,

$$P_{2m}(s_l, s_k) = Q\left(\frac{2dE}{N_0}\right) \tag{2.49}$$

Under the assumption that all codes are equally probable, it is equally likely that the vector s_l is any of the 2^n vertices of the hypercube. Further, s_l and s_k are statistically independent. Hence, the probability that s_l and s_k differ in exactly d places is

$$P(d) = \left(\frac{1}{2}\right)^n \binom{n}{d} \tag{2.50}$$

The expected value of $P_{2m}(\boldsymbol{s}_l, \boldsymbol{s}_k)$ over the ensemble of codes is then given by

$$P_2(\boldsymbol{s}_l, \boldsymbol{s}_k) = \sum_{d=0}^{n} P(d)\, Q\left(\frac{2dE}{N_0}\right) = \left(\frac{1}{2^n}\right)\sum_{d=0}^{n}\binom{n}{d} Q\left(\frac{2dE}{N_0}\right) \tag{2.51}$$

Using the following upper bound

$$Q\left(\frac{2dE}{N_0}\right) < e^{-\frac{dE}{N_0}}, \tag{2.52}$$

we obtain

$$P_2(\boldsymbol{s}_l, \boldsymbol{s}_k) \le \left(\frac{1}{2^n}\right)\sum_{d=0}^{n}\binom{n}{d} e^{\frac{-dE}{N_0}} \le \left[\frac{1}{2}\left(1 + e^{\frac{-E}{N_0}}\right)\right]^n \tag{2.53}$$

From Eqs. (2.46) and (2.53) we obtain

$$\overline{P_e(\boldsymbol{X}_k)} \le \sum_{\substack{l=1 \\ l \ne k}}^{M} \overline{P_2(\boldsymbol{s}_l, \boldsymbol{s}_k)} = (M-1)\left[\frac{1}{2}\left(1 + e^{\frac{-E}{N_0}}\right)\right]^n < M\left[\frac{1}{2}\left(1 + e^{\frac{-E}{N_0}}\right)\right]^n \tag{2.54}$$

Recall that we need an upper bound on $\overline{P}_e$, the average error probability. To obtain $\overline{P}_e$ we average $\overline{P_e(\boldsymbol{X}_k)}$ over all possible k-bit information sequences. Thus,

$$\overline{P}_e = \sum_k \overline{P_e(\boldsymbol{X}_k)} P(\boldsymbol{X}_k) < M\left[\frac{1}{2}\left(1 + e^{\frac{-E}{N_0}}\right)\right]^n \sum_k P(\boldsymbol{X}_k) < M\left[\frac{1}{2}\left(1 + e^{\frac{-E}{N_0}}\right)\right]^n \tag{2.55}$$

We now define a new parameter as follows.

Definition 2.4 The **Cutoff Rate** R_0 is defined as

$$R_0 = \log_2 \frac{2}{1 + e^{\frac{-E}{N_0}}} = 1 - \log_2\left(1 + e^{\frac{-E}{N_0}}\right). \tag{2.56}$$

The cutoff rate has the units of bits/dimension. Observe that $0 \le R_0 \le 1$. The plot of R_0 with respect to SNR per dimension is given in Fig. 2.8.

The Eq. (2.55) can now be written succinctly as

$$\overline{P}_e < M 2^{-nR_0} = 2^{-RT} 2^{-nR_0} \tag{2.57}$$

Substituting $n = DT$, we obtain

$$\overline{P}_e < 2^{-T(DR_0 - R)} \tag{2.58}$$

If we substitute $T = n/D$, we obtain

$$\overline{P}_e < 2^{-n(R_0 - R/D)} \tag{2.59}$$

Fig. 2.8 *Cutoff Rate, R_0, Versus the SNR (in dB) Per Dimension.*

Observe that

$$\frac{R}{D} = \frac{R}{n/T} = \frac{RT}{n} = \frac{k}{n} = R_c. \tag{2.60}$$

Here, R_c represents the code rate. Hence the average error probability can be written in the following instructive form

$$\overline{P}_e < 2^{-n(R_0 - R_c)} \tag{2.61}$$

From the above equation we can conclude the following.

(i) For $R_c < R_0$ the average probability of error $\overline{P}_e \to 0$ as $n \to \infty$. Since by choosing large values of n, $\overline{P}_e$ can be made arbitrarily small, there exist good codes in the ensemble which have the probability of error less than $\overline{P}_e$.

(ii) We observe the fact that $\overline{P}_e$ is the ensemble average. Therefore, if a code is selected at random, the probability is that its error $\overline{P}_e > \alpha\, \overline{P}_e$ is less than $1/\alpha$. This implies that there are no more than 10% of the codes that have an error probability that exceeds $10\overline{P}_e$. Thus, there are many good codes.

(iii) The codes whose probability of error exceed $\overline{P}_e$ are not always bad codes. The probability of error of these codes may be reduced by increasing the dimensionality, n.

For binary coded signals, the cutoff rate, R_0, saturates at 1 bit/dimension for large values of $\frac{E}{N_0}$, say >10. Thus, to achieve lower probabilities of error one must reduce the code rate, R_c.

Alternately, very large block lengths have to be used. This is not an efficient approach. So, binary codes become inefficient at high SNRs. For high SNR scenarios, non-binary coded signal sets should be used to achieve an increase in the number of bits per dimension. Multiple-amplitude coded signal sets can be easily constructed from non-binary codes by mapping each code element into one of the possible amplitude levels (e.g. Pulse Amplitude Modulation). For random codes using M-ary multi-amplitude signals, it was shown by Shannon (in 1959) that

$$R_0^* = \frac{1}{2}\left[1 + \frac{E}{N_0} - \sqrt{1 + \left(\frac{E}{N_0}\right)^2}\right]\log_2 e + \frac{1}{2}\log_2\left[\frac{1}{2}\left(1 + \sqrt{1 + \left(\frac{E}{N_0}\right)^2}\right)\right] \tag{2.62}$$

Let us now relate the cutoff rate R_0^* to the capacity of the AWGN channel, which is given by

$$C = W\log_2\left(1 + \frac{P}{N_0 W}\right) \text{ bits per second} \tag{2.63}$$

The energy per code bit is equal to

$$E = \frac{PT}{n} \tag{2.64}$$

Recall that from the sampling theorem, a signal of bandwidth W may be represented by samples taken at a rate $2W$. Thus, in the time interval of length T there are $n = 2WT$ samples. Therefore, we may write $D = \frac{n}{T} = 2W$. Hence,

$$P = \frac{nE}{T} = DE. \tag{2.65}$$

Define normalized capacity, $C_n = \frac{C}{2W} = \frac{C}{D}$ and substitute for W and P in (2.63) to obtain

$$\begin{aligned} C_n &= \left(\frac{1}{2}\right)\log_2\left(1 + 2\frac{E}{N_0}\right) \\ &= \left(\frac{1}{2}\right)\log_2(1 + 2R_c\gamma_b) \end{aligned} \tag{2.66}$$

where γ_b is the SNR per bit. The normalized capacity, C_n and cutoff rate, R_0^*, are plotted in Fig. 2.9. From the figure we can conclude the following:

(i) $R_0^* < C_n$ for all values of $\frac{E}{N_0}$. This is expected because C_n is the ultimate limit on the transmission rate R/D.

(ii) For smaller values of the difference between C_n and R_0^* is approximately 3 dB. This means that randomly selected, average power limited, multi-amplitude signals yield R_0^* within 3 dB of channel capacity.

Fig. 2.9 *The Normalized Capacity, C_n and Cutoff Rate, R_0^*, for an AWGN Channel.*

2.8 CONCLUDING REMARKS

Pioneering work in the area of channel capacity was done by Shannon in 1948. Shannon's second theorem was indeed a surprising result at the time of its publication. It claimed that the probability of error for a BSC could be made as small as desired provided the code rate was less than the channel capacity. This theorem paved the way for a systematic study of reliable communication over unreliable (noisy) channels. Shannon's third theorem, the **Information Capacity Theorem**, is one of the most remarkable results in information theory. It gives a relation between the channel bandwidth, the signal to noise ratio and the channel capacity. Additional work was carried out in the 1950s and 1960s by Gilbert, Gallager, Wyner, Forney and Viterbi to name some of the prominent contributors.

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 channels, including fading channels encountered in wireless communications.

SUMMARY

- The conditional probability $P(y_i \mid x_j)$ is called the channel transition probability and is denoted by p_{ji}. The conditional probabilities $\{P(y_i \mid x_j)\}$ that characterize a DMC can be arranged in the matrix form $\boldsymbol{P} = [p_{ji}]$. $\boldsymbol{P}$ 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 maximization is over all possible input probabilities. That is,

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

- 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 Coding Theorem**.
- For $\frac{H(X)}{T_s} \le \frac{C}{T_c}$, 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 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 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 $\bar{P}_e < 2^{-n(R_0 - R_c)}$. For $R_c < R_0$ the average probability of error $\bar{P}_e \to 0$ 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 $P(X = 0 \mid Y = 0)$ and $P(X = 1 \mid Y = 1)$

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.
(a) Find C_A the capacity of channel A.
(b) Find C_B the capacity of channel B.
(c) Next, cascade the two channels and determine the combined capacity C_{AB}.
(d) 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.

Fig. 2.13

2.5 (a) A telephone channel has a bandwidth of 3000 Hz and the SNR = 20 dB. Determine the channel capacity.

(b) If the SNR is increased to 25 dB, determine the capacity.

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

Fig. 2.14

2.7 Suppose a TV displays 30 frames/second. There are approximately 2×10^5 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**).

2.8 Consider the Z channel shown in Fig. 2.15.

(a) Find the input probabilities that result in capacity.

(b) If N such channels are cascaded, show that the combined channel can be represented by an equivalent Z channel with the channel transition probability p^N.

(c) What is the capacity of the combined channel as $N \to \infty$?

Fig. 2.15

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

(a) Find the cutoff rate, R_0.

(b) We want to design a code which results in an average probability of error, $P_e < 10^{-6}$. What is the best code rate we can achieve?

(c) What will be the dimensionality, n, of this code?

(d) Repeat the earlier parts (a), (b) and (c) for an SNR = 5dB. Compare the results.

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

(b) 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 repitition code of rate $1/n$, where n is an odd integer. Develop a decoder for the repitition 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).

PART II

Information Theory, Coding and Cryptography

Error Control Coding (Channel Coding)

3

Linear Block Codes for Error Correction

Mathematics is an interesting intellectual sport but it should not be allowed to stand in the way of obtaining sensible information about physical processes.

Richard W. Hamming

3.1 INTRODUCTION TO ERROR CORRECTING CODES

In this *age of information*, there is increasing need not only for speed, but also for accuracy in the storage, retrieval, and transmission of data. The channels over which messages are transmitted are often imperfect. Machines do make errors, and their non-man-made mistakes can turn otherwise flawless programming into worthless, even dangerous, trash. Just as architects design buildings that will stand even through an earthquake, their computer counterparts have come up with sophisticated techniques capable of counteracting digital manifestations of Murphy's Law ("If anything can go wrong, it will go"). **Error Correcting Codes** are a kind of safety net–the mathematical insurance against the vagaries of an imperfect digital world.

Error Correcting Codes, as the name suggests, are used for correcting errors when messages are transmitted over a noisy channel or stored data is retrieved. The physical medium through which the messages are transmitted is called a channel (e.g. a telephone line, a satellite link, a wireless channel used for mobile communications etc.). Different kinds of channels are

prone to different kinds of noise, which corrupt the data being transmitted. The noise could be caused by lightning, human errors, equipment malfunction, voltage surge etc. Because these error correcting codes try to overcome the detrimental effects of noise in the channel, the encoding procedure is also called *Channel Coding*. Error control codes are also used for accurate transfer of information from one place to another, for example storing data and reading it from a compact disc (CD). In this case, the error could be due to a scratch on the surface of the CD. The error correcting coding scheme will try to recover the original data from the corrupted one.

The basic idea behind error correcting codes is to add some redundancy in the form of extra symbol to a message prior to its transmission through a noisy channel. This redundancy is added in a controlled manner. The encoded message when transmitted might be corrupted by noise in the channel. At the receiver, the original message can be recovered from the corrupted one if the number of errors are within the limit for which the code has been designed. The block diagram of a digital communication system is illustrated in Fig. 3.1. Note that the most important block in the figure is that of noise, without which there will be no need for the channel encoder.

Example 3.1 Let us see how redundancy combats the effects of noise. The normal language that we use to communicate (say, English) has a lot of redundancy built into it. Consider the following sentence.

CODNG THEORY IS AN INTRSTNG SUBJCT.

As we can see, there are a number of errors in this sentence. However, due to familiarity with the language we may guess the original text to have read:

CODING THEORY IS AN INTERESTING SUBJECT.

What we have just used is an error correcting strategy that makes use of the in-built redundancy in English language to reconstruct the original message from the corrupted one.

Fig. 3.1 *Block Diagram (and the principle) of a Digital Communication System. Here the Source Coder/Decoder Block has not been shown.*

The objectives of a good error control coding scheme are

(i) error correcting capability in terms of the number of errors that it can rectify
(ii) fast and efficient encoding of the message,
(iii) fast and efficient decoding of the received message
(iv) maximum transfer of information bits per unit time (i.e., fewer overheads in terms of redundancy).

The first objective is the primary one. In order to increase the error correcting capability of a coding scheme one must introduce more redundancies. However, increased redundancy leads to a slower rate of transfer of the actual information. Thus the objectives (i) and (iv) are not totally compatible. Also, as the coding strategies become more complicated for correcting larger number of errors, the objectives (ii) and (iii) also become difficult to achieve.

In this chapter, we shall first learn about the basic definitions of error control coding. These definitions, as we shall see, would be used throughout this book. The concept of **Linear Block Codes** will then be introduced. **Linear Block Codes** form a very large class of useful codes. We will see that it is very easy to work with the matrix description of these codes. In the later part of this chapter, we will learn how to efficiently decode these **Linear Block Codes**. Finally, the notion of perfect codes and optimal linear codes will be introduced.

3.2 BASIC DEFINITIONS

Given here are some basic definitions, which will be frequently used here as well as in the later chapters.

Definition 3.1 A **Word** is a sequence of symbols.

Definition 3.2 A **Code** is a set of vectors called **Codewords**.

Definition 3.3 The **Hamming Weight** of a **Codeword** (or any vector) is equal to the number of nonzero elements in the codeword. The Hamming Weight of a codeword c is denoted by $w(c)$. The **Hamming Distance** between two codewords is the number of places the codewords differ. The Hamming Distance between two codewords c_1 and c_2 is denoted by $d(c_1, c_2)$. It is easy to see that $d(c_1, c_2) = w(c_1 - c_2)$.

Example 3.2 Consider a code C with two code words = {0100, 1111} with Hamming Weight $w(0100) = 1$ and $w(1111) = 4$. The Hamming Distance between the two codewords is 3 because they differ at the 1st, 3rd and 4th places. Observe that $w(0100 - 1111) = w(1011) = 3 = d(0100, 1111)$.

Example 3.3 For the code $C = \{01234, 43210\}$, the Hamming Weight of each codeword is 4 and the Hamming Distance between the codewords is 4 (because only the 3rd component of the two codewords are identical while they differ at 4 places).

Definition 3.4 A **Block Code** consists of a set of fixed length codewords. The fixed length of these codewords is called the **Block Length** and is typically denoted by n. Thus, a code of blocklength n consists of a set of codewords having n **components**.

A block code of size M defined over an alphabet with q symbols is a set of M q-ary sequences, each of length n. In the special case that $q = 2$, the symbols are called bits and the code is said to be a binary code. Usually, $M = q^k$ for some integer k, and we call such a code an (n, k) code.

Example 3.4 The code $C = \{00000, 10100, 11110, 11001\}$ is a block code of block length equal to 5. This code can be used to represent two bit binary numbers as follows

Uncoded bits	Codewords
00	00000
01	10100
10	11110
11	11001

Here $M = 4$, $k = 2$ and $n = 5$. Suppose we have to transmit a sequence of 1's and 0's using the above coding scheme. Let's say that the sequence to be encoded is 1 0 0 1 0 1 0 0 1 1 ... The first step is to break the sequence in groups of two bits (because we want to encode two bits at a time). So we partition as follows

10 01 01 00 11 ...

Next, replace each block by its corresponding codeword.

11110 10100 10100 00000 11001 ...

Thus 5 bits (coded) are sent for every 2 bits of uncoded message. It should be observed that for every 2 bits of information we are sending 3 extra bits (redundancy).

Definition 3.5 The **Code Rate** of an (n, k) code is defined as the ratio (k/n), and denotes the fraction of the codeword that consists of the information symbols.

Code rate is always less than unity. The smaller the code rate, the greater the redundancy, i.e., more of redundant symbols are present *per information symbol* in a codeword. A code with greater redundancy has the potential to detect and correct more of symbols in error, but reduces the actual rate of transmission of information.

Definition 3.6 The **minimum distance** of a code is the minimum Hamming distance between any two codewords. If the code C consists of the set of codewords $\{c_i, i = 0, 1, ..., M-1\}$ then the minimum distance of the code is given by $d^* = \min d(c_i, c_j)$, $i \neq j$. An (n, k) code with minimum distance d^* is sometimes denoted by (n, k, d^*).

Definition 3.7 The **minimum weight** of a code is the smallest weight of any non-zero codeword, and is denoted by w^*.

Theorem 3.1 For a linear code the minimum distance is equal to the minimum weight of the code, i.e., $d^* = w^*$.

Intuitive proof: The distance d_{ij} between any two codewords c_i and c_j is simply the weight of the codeword formed by $c_i - c_j$. Since the code is linear, the difference between two codewords results in another valid codeword. Thus, the minimum weight of a non-zero codeword will reflect the minimum distance of the code.

Definition 3.8 A **linear code** has the following properties:

(i) The sum of two codewords belonging to the code is also a codeword belonging to the code.

(ii) The all-zero word is always a codeword.

(iii) The minimum Hamming distance between two codewords of a linear code is equal to the minimum weight of any non-zero codeword, i.e., $d^* = w^*$.

Note that if the sum of two codewords is another codeword, the difference of two codewords will also yield a valid codeword. For example, if c_1, c_2 and c_3 are valid codewords such that $c_1 + c_2 = c_3$ then $c_3 - c_1 = c_2$. Hence it is obvious that the all-zero codeword must always be a valid codeword for a linear block code (self-subtraction of a codeword).

Example 3.5 The code $C = \{0000, 1010, 0101, 1111\}$ is a linear block code of block length $n = 4$. Observe that all the ten possible sums of the codewords

0000 + 0000 = 0000, 0000 + 1010 = 1010, 0000 + 0101 = 0101,

0000 + 1111 = 1111, 1010 + 1010 = 0000, 1010 + 0101 = 1111,

1010 + 1111 = 0101, 0101 + 0101 = 0000, 0101 + 1111 = 1010 and

1111 + 1111 = 0000

are in C and the all-zero codeword is in C. The minimum distance of this code is $d^* = 2$. In order to verify the minimum distance of this linear w code we can determine the distance between all pairs of codewords (which is $\binom{4}{2} = 6$ in number):

d (0000, 1010) = 2, d (0000, 0101) = 2, d (0000, 1111) = 4

d (1010, 0101) = 4, d (1010, 1111) = 2, d (0101, 1111) = 2

We observe that the minimum distance of this code is 2.

Note that the code given in Example 3.4 is not linear because 1010 + 1111 = 0101, which is not a valid codeword. Even though the all-zero word is a valid codeword, it does not guarantee linearity. **The presence of an all-zero codeword is thus a necessary but not a sufficient condition for linearity.**

In order to make the error correcting codes easier to use, understand and analyze, it is helpful to impose some basic algebraic structure on them. As we shall soon see, it is useful to have an alphabet werein it is easy to carry out basic mathematical operations such as add, subtract, multiply and divide.

Definition 3.9 A field F is a set of elements with two operations + (addition) and . (multiplication) satisfying the following properties

(i) F is closed under + and ., i.e., $a + b$ and $a \cdot b$ are in F if a and b are in F.

For all a, b and c in F, the following hold:

(ii) Commutative laws: $a + b = b + a$, $a \cdot b = b.a$

(iii) Associative laws: $(a + b) + c = a + (b + c)$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

(iv) Distributive law: $a \cdot (b + c) = a \cdot b + a \cdot c$

Further, identity elements 0 and 1 must exist in F satisfying:

(v) $a + 0 = a$

(vi) $a \cdot 1 = a$

(vii) For any a in F, there exists an additive inverse $(-a)$ such that $a + (-a) = 0$.

(viii) For any a in F, there exists an multiplicative inverse (a^{-1}) such that $a \cdot a^{-1} = 1$.

The above properties are true for fields with both finite as well as infinite elements. A field with a finite number of elements (say, q) is called a Galois Field (pronounced Galva Field) and is denoted by $GF(q)$. If only the first seven properties are satisfied, then it is called a **ring**.

Example 3.6 Consider GF (4) with 4 elements {0, 1, 2, 3}. The addition and multiplication tables for $GF(4)$ are

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

It should be noted here that the addition in $GF(4)$ is not modulo 4 addition.

Let us define a vector space, $GF(q)^n$, which is a set of n-tuples of elements from $GF(q)$. Linear block codes can be looked upon as a set of n-tuples (vectors of length n) over $GF(q)$ such that the sum of two codewords is also a codeword, and the product of any codeword by a field element is a codeword. Thus, a linear block code is a subspace of $GF(q)^n$.

Let S be a set of vectors of length n whose components are defined over $GF(q)$. The set of all linear combinations of the vectors of S is called the linear span of S and is denoted by $<S>$. The linear span is thus a subspace of $GF(q)^n$, generated by S. Given any subset S of $GF(q)^n$, it is possible to obtain a linear code $C = <S>$ generated by S, consisting of precisely the following codewords:

(i) all-zero word,

(ii) all words in S,

(iii) all linear combinations of two or more words in S.

Example 3.7 Let S = {1100, 0100, 0011}. All possible linear combinations of S are 1100 + 0100 = 1000, 1100 + 0011 = 1111, 0100 + 0011 = 0111, 1100 + 0100 + 0011 = 1011.

Therefore, $C = <S>$ = {0000, 1100, 0100, 0011, 1000, 1111, 0111, 1011}. The minimum distance of this code is $w(0100) = 1$.

Example 3.8 Let S = {12, 21} defined over $GF(3)$. The addition and multiplication tables of field $GF(3)$ = {0, 1, 2} are given by:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

.	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

All possible linear combinations of 12 and 21 are:

12 + 21 = 00, 12 + 2(21) = 21, 2(12) + 21 = 12.

Therefore, $C = <S>$ = {00, 12, 21, 00, 21, 12} = {00, 12, 21}.

3.3 MATRIX DESCRIPTION OF LINEAR BLOCK CODES

As we have observed earlier, any code C is a subspace of $GF(q)^n$. Any set of basis vectors can be used to generate the code space. We can, therefore, define a generator matrix, G, the rows of which form the basis vectors of the subspace. The rows of G will be linearly independent. Thus, a linear combination of the rows can be used to generate the codewords of C. The generator matrix will be a $k \times n$ matrix with rank k. Since the choice of the basis vectors is not unique, *the generator matrix is not unique for a given linear code.*

The generator matrix converts (encodes) a vector of length k to a vector of length n. Let the input vector (uncoded symbols) be represented by $\boldsymbol{i}$. The coded symbols will be given by

$$\boldsymbol{c} = \boldsymbol{iG} \tag{3.1}$$

where $\boldsymbol{c}$ is called the codeword and $\boldsymbol{i}$ is called the information word.

The generator matrix provides a concise and efficient way of representing a linear block code. The $n \times k$ matrix can generate q^k codewords. Thus, instead of having a large look-up table of q^k codewords, one can simply have a generator matrix. This provides an enormous saving in storage space for large codes. For example, for the binary (46, 24) code the total number of codewords are $2^{24} = 1{,}777{,}216$ and the size of the lookup table of codewords will be $n \times 2^k =$ 771,751,936 bits. On the other hand if we use a generator matrix, the total storage requirement would be $n \times k = 46 \times 24 = 1104$ bits.

Example 3.9 Consider a generator matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{c}_1 = [0\ 0]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [0\ 0\ 0], \qquad \boldsymbol{c}_2 = [0\ 1]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [0\ 1\ 0]$$

$$\boldsymbol{c}_3 = [1\ 0]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [1\ 0\ 1], \qquad \boldsymbol{c}_4 = [1\ 1]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [1\ 1\ 1]$$

Therefore, this generator matrix generates the code $\boldsymbol{C} = \{000, 010, 101, 111\}$.

3.4 EQUIVALENT CODES

Definition 3.10 A **permutation** of a set $\boldsymbol{S} = \{x_1, x_2, \ldots, x_n\}$ is a one to one mapping from $\boldsymbol{S}$ to itself. A permutation can be denoted as follows

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ \downarrow & \downarrow & \cdots & \downarrow \\ f(x_1) & f(x_2) & \cdots & f(x_n) \end{pmatrix} \tag{3.2}$$

Definition 3.11 Two q-ary codes are called **equivalent** if one can be obtained from the other by one or both operations listed below:

(i) permutation of the symbols appearing in a fixed position,
(ii) permutation of the positions of the code.

Suppose a code containing M codewords are displayed in the form of an $M \times n$ matrix, where the rows represent the codewords. The operation (i) corresponds to the re-labelling of the symbols appearing in a given column, and the operation (ii) represents the rearrangements of the colums of the matrix.

Example 3.10 Consider the ternary code (a code whose components $\in \{0, 1, 2\}$) of blocklength 3

$$\boldsymbol{C} = \begin{cases} 2 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \end{cases}$$

If we apply the permutation $0 \to 2$, $2 \to 1$, $1 \to 0$ to column 2 and $1 \to 2$, $0 \to 1$, $2 \to 0$ to column 3 we obtain

$$\boldsymbol{C1} = \begin{cases} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{cases}$$

The code $\boldsymbol{C1}$ is equivalent to a repetition code of length 3.

Note that the original code is not linear, but is equivalent to a linear code.

Definition 3.12 Two linear q-ary codes are called **equivalent** if one can be obtained from the other by one or both operations listed below:

(i) multiplication of the components by a non-zero scalar,
(ii) permutation of the positions of the code.

Note that in Definition 3.11 we have defined equivalent codes that are not necessarily linear.

Theorem 3.2 Two $k \times n$ matrices generate equivalent linear (n, k) codes over $GF(q)$ if one matrix can be obtained from the other by a sequence of the following operations:

(i) Permutation of rows
(ii) Multiplication of a row by a non scalar
(iii) Addition of a scalar multiple of one row to another
(iv) Permutation of columns
(v) Multiplication of any column by a non-zero scalar.

Proof The first three operations (which are just row operations) preserve the linear independence of the rows of the generator matrix. The operations merely modify the basis. The last two operations (which are column operations) convert the matrix to one which will produce an equivalent code.

Theorem 3.3 A generator matrix can be reduced to its **systematic form** (also called the standard form of the generator matrix) of the type $\boldsymbol{G} = [\,\boldsymbol{I} \mid \boldsymbol{P}]$ where $\boldsymbol{I}$ is a $k \times k$ identity matrix and $\boldsymbol{P}$ is a $k \times (n-k)$ matrix.

Proof The k rows of any generator matrix (of size $k \times n$) are linearly independent. Hence, by performing elementary row operations and column permutations it is possible to obtain an equivalent generator matrix in a row echelon form. This matrix will be of the form $[\boldsymbol{I} \mid \boldsymbol{P}]$.

Example 3.11 Consider the generator matrix of a (4, 3) code over $GF(3)$:

$$\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 2 & 2 & 1 \end{bmatrix}$$

Let us represent the ith row by r_i and the jth column by r_j. Upon replacing r_3 by $r_3 - r_1 - r_2$ we get (note that in $GF(3)$, $-1 = 2$ and $-2 = 1$ because $1 + 2 = 0$, see table in Example 3.6)

$$\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Next we replace r_1 by $r_1 - r_3$ to obtain

$$\boldsymbol{G} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Finally, shifting $c_4 \to c_1$, $c_1 \to c_2$, $c_2 \to c_3$ and $c_3 \to c_4$ we obtain the standard form of the generator matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

3.5 PARITY CHECK MATRIX

One of the objectives of a good code design is to have fast and efficient encoding and decoding methodologies. So far we have dealt with the efficient generation of linear block codes using a generator matrix. Codewords are obtained simply by multiplying the input vector (uncoded word) by the generator matrix. Is it possible to detect a valid codeword using a similar concept? The answer is yes, and such a matrix is called the **Parity Check Matrix**, $\boldsymbol{H}$, for the given code. For a parity check matrix,

$$\boldsymbol{c}\boldsymbol{H}^T = \boldsymbol{0} \tag{3.3}$$

where $\boldsymbol{c}$ is a valid codeword. Since $\boldsymbol{c} = \boldsymbol{i}\boldsymbol{G}$, therefore, $\boldsymbol{i}\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$. For this to hold true for all valid informat words we must have

$$\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0} \tag{3.4}$$

The size of the parity check matrix is $(n-k) \times n$. A parity check matrix provides a simple method of *detecting* whether an error has occurred or not. If the multiplication of the received word (at the receiver) with the transpose of $\boldsymbol{H}$ yields a non-zero vector, it implies that an error has occurred. This methodology, however, will fail if the errors in the transmitted codeword exceed the number of errors for which the coding scheme is designed. We shall soon find out that the non-zero product of $\boldsymbol{c}\boldsymbol{H}^T$ might help us not only to detect but also to correct the errors under some conditions.

Suppose the generator matrix is represented in its systematic form $\boldsymbol{G} = [\,\boldsymbol{I} \mid \boldsymbol{P}]$. The matrix $\boldsymbol{P}$ is called the **Coefficient Matrix**. Then the parity check matrix will be defined as

$$\boldsymbol{H} = [\,-\boldsymbol{P}^T \mid \boldsymbol{I}], \tag{3.5}$$

where $\boldsymbol{P}^T$ represents the transpose of matrix $\boldsymbol{P}$. This is because

$$\boldsymbol{G}\boldsymbol{H}^T = [\,\boldsymbol{I} \mid \boldsymbol{P}] \begin{bmatrix} -\boldsymbol{P} \\ \boldsymbol{I} \end{bmatrix} = \boldsymbol{0}. \tag{3.6}$$

Since the choice of a generator matrix is not unique for a code, *the parity check matrix will not be unique either.* Given a generator matrix $\boldsymbol{G}$, we can determine the corresponding parity check matrix and *vice versa.* Thus the parity check matrix $\boldsymbol{H}$ can be used to specify the code completely.

From Eq. (3.3) we observe that the vector $\boldsymbol{c}$ must have 1's in such positions that the corresponding rows of $\boldsymbol{H}^T$ add up to the zero vector $\boldsymbol{0}$. Now, we know that the number of 1's in a codeword pertains to its Hamming weight. *Hence, the minimum distance d^* of a linear block code is given by the minimum number of rows of $\boldsymbol{H}^T$ (or, the columns of $\boldsymbol{H}$) whose sum is equal to the zero vector.*

Example 3.12 For a (7, 4) linear block code the generator matrix is given by

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix},$$

the matrix P is given by $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ and P^T is given by $\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$. Observing the fact that $-1 = 1$ for the case of binary, we can write the parity check matrix as

$$H = [-P^T \mid I]$$

$$= \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that the columns 1, 5 and 7 of the parity check matrix, H, add up to the zero vector. Hence, for this code, $d^* = 3$.

Theorem 3.4 The code C contains a nonzero codeword of Hamming weight w or less if and only if a linearly dependent set of w columns of H exist.

Proof Consider a codeword $c \in C$. Let the weight of c be w which implies that there are w non-zero components and $(n - w)$ zero components in c. If we throw away the w zero components, then from the relation $CH^T = 0$ we can conclude that w columns of H are linearly dependent.

Conversely, if H has w linearly dependent columns, then a linear combination of at most w columns is zero. These w non-zero coefficients would define a codeword of weight w or less that satisfies $CH^T = 0$.

Definition 3.13 An (n, k) **systematic code** is one in which the first k symbols of the codeword of block length n are the information symbols themselves (i.e., the uncoded vector) and the remainder the $(n - k)$ symbols form the **parity symbols.**

Example 3.13 The following is a (5, 2) systematic code over $GF(3)$

S.No.	Information Symbols (k = 2)	Codewords (n = 5)
1.	00	00 000
2.	01	01 121
3.	02	02 220
4.	10	10 012
5.	11	11 221
6.	12	12 210
7.	20	20 020
8.	21	21 100
9.	22	22 212

Note that the total number of codewords is $3^k = 3^2 = 9$. Each codeword begins with the information symbols, and has three parity symbols at the end. The parity symbols for the information word 01 are 121 in the above table. A generator matrix in the systematic form (standard form) will generate a systematic code.

Theorem 3.5 The minimum distance (minimum weight) of an (n, k) linear code is bounded as follows

$$d^* \le n - k + 1 \tag{3.7}$$

This is known as the **Singleton Bound.**

Proof We can reduce all linear block codes to their equivalent systematic forms. A systematic code can have one information symbol and $(n - k)$ parity symbols. At most all the parity symbols can be non-zero, resulting in the total weight of the codeword to be $(n - k) + 1$. Thus the weight of no codeword can exceed $n - k + 1$ giving the following definition of a maximum distance code.

Definition 3.14 A **Maximum Distance Code** satisfies $d^* = n - k + 1$.

Having familiarized ourselves with the concept of minimum distance of a linear code, we shall now explore how this minimum distance is related to the total number of errors the code can detect and possibly correct. So we move over to the receiver end and take a look at the methods of decoding a linear block code.

3.6 DECODING OF A LINEAR BLOCK CODE

The basic objective of channel coding is to detect and correct errors when messages are transmitted over a noisy channel. The noise in the channel randomly transforms some of the

symbols of the transmitted codeword into some other symbols. If the noise, for example, changes just one of the symbols in the transmitted codeword, the erroneous codeword will be at a Hamming distance of one from the original codeword. If the noise transforms t symbols (that is, t symbols in the codeword are in error), the Hamming distance of the received word will be at a Hamming distance of t from the originally transmitted codeword. Given a code, how many errors can it detect and how many can it correct? Let us first look at the detection problem.

An error will be detected as long as it does not transform one codeword into another valid codeword. If the minimum distance between the codewords is d^*, the weight of the error pattern must be d^* or more to cause a transformation of one codeword to another. Therefore, an (n, k, d^*) code will detect at least all nonzero error patterns of weight less than or equal to $(d^* - 1)$.

Moreover, there is at least one error pattern of weight d^* which will not be detected. This corresponds to the two codewords that are the closest. It may be possible that some error patterns of weight d^* or more are detected, but *all* error patterns of weight d^* will not be detected.

Example 3.14 For the code $C_1 = \{000, 111\}$ the minimum distance is 3. Therefore error patterns of weight 2 or 1 can be detected. This means that any error pattern belonging to the set {011, 101, 110, 001, 010, 100} will be detected by this code.

Next consider the code $C_2 = \{001, 110, 101\}$ with $d^* = 1$. Nothing can be said regarding how many errors this code can detect because $d^* - 1 = 0$. However, the error pattern 010 of weight 1 can be detected by this code. But it cannot detect all error patterns with weight one, e.g., the error vector 100 cannot be detected.

Next let us look at the problem of error correction. The objective is to make the best possible guess regarding the originally transmitted codeword on the basis of the received word. What would be a smart decoding strategy? Since only one of the valid codewords must have been transmitted, it is logical to conclude that a valid codeword nearest (in terms of Hamming distance) to the received word must have been actually transmitted. In other words, the codeword which resembles the received word most is assumed to be the one that was sent. This strategy is called the **Nearest Neighbour Decoding**, as we are picking the codeword nearest to the received word in terms of the Hamming distance.

It may be possible that more than one codeword is at the same Hamming distance from the received word. In that case the receiver can do one of the following:

(i) It can pick one of the equally distant neighbours randomly, or
(ii) request the transmitter to re-transmit.

To ensure that the received word (that has at most t errors) is closest to the original codeword, and farther from all other codewords, we must put the following condition on the minimum distance of the code

$$d^* \le 2t + 1 \tag{3.8}$$

Graphically, the condition for correcting t errors or less can be visualized from Fig. 3.2. Consider the space of all q-ary n-tuples. Every q-ary vector of length n can be represented as a point in this space. Every codeword can thus be depicted as a point in this space, and all words at a Hamming distance of t or less would lie within the sphere centred at the codeword and with a radius of t. If the minimum distance of the code is d^*, and the condition $d^* \ge 2t + 1$ holds good, then none of these spheres would intersect. Any received vector (which is just a point) within a specific sphere will be closest to its centre (which represents a codeword) than any other codeword. We will call the spheres associated with each codeword its **Decoding Sphere**. Hence it is possible to decode the received vector using the 'nearest neighbour' method without ambiguity.

Fig. 3.2 *Decoding Spheres.*

Figure 3.2 shows words within the sphere of radius t and centred at c_1 will be decoded as c_1. For unambiguous decoding $d^* \ge 2t + 1$.

The condition $d^* \ge 2t + 1$ takes care of the worst case scenario. It may be possible, however, that the above condition is not met but it is still feasible to correct t errors as illustrated in the following example.

Example 3.15 Consider the code $C = \{00000, 01010, 10101, 11111\}$. The minimum distance $d^* = 2$. Suppose the codeword 11111 was transmitted and the received word is 11110, i.e., $t = 1$ (one error has occurred in the fifth component). Now,

$$d(11110, 00000) = 4,\ d(11110, 01010) = 2,$$
$$d(11110, 10101) = 3,\ d(11110, 11111) = 1.$$

Using the nearest neighbour decoding we can conclude that 11111 was transmitted. Even though a single error correction ($t = 1$) was done in this case, $d^* < 2t + 1 = 3$. So it is possible to correct

errors even when $d^* \geq 2t + 1$. However, in many cases a single error correction may not be possible with this code. For example, if 00000 was sent and 01000 was received,

$$d\,(01000, 00000) = 1,\ d\,(01000, 01010) = 1,$$
$$d\,(01000, 10101) = 4,\ d\,(01000, 11111) = 4.$$

In this case there cannot be a clear cut decision, and a coin will have to be flipped!

Definition 3.15 An **Incomplete Decoder** decodes only those received codewords that are clearly closest to one of the codewords. In the case of ambiguity, the decoder declares that the received word is unrecognizable. The receiver is then requested to re-transmit. A **Complete Decoder** decodes every received word, i.e., it tries to map every received word to some codeword, even if it has to make a guess. Example 3.16 was that of a Complete Decoder. Such decoders may be used when it is better to have a good guess rather than to have no guess at all. Most of the real life decoders are incomplete decoders. Usually they send a message back to the transmitter requesting them to re-transmit.

Definition 3.16 A receiver declares that an **erasure** has occurred (i.e., a received symbol has been erased) when the symbol is received ambiguously, or the presence of an interference is detected during reception.

Example 3.16 Consider a binary **Pulse Amplitude Modulation** (PAM) **Scheme** where 1 is represented by five volts and 0 is represented by zero volts. The noise margin is one volt, which implies that at the receiver:

if the received voltage is between 4 volts and 5 volts $\Rightarrow$ the bit sent is 1,

if the received voltage is between 0 volt and 1 volt $\Rightarrow$ the bit sent is 0,

if the received voltage is between 1 volt and 4 volts $\Rightarrow$ an erasure has occurred.

Thus if the receiver received 2.9 volts during a bit interval, it will declare that an erasure has occurred.

A channel can be prone both to errors and erasures. If in such a channel t errors and r erasures occur, the error correcting scheme should be able to compensate for the erasures as well as correct the errors. If r erasures occur, the minimum distance of the code will become $d^* - r$ in the worst case. This is because, the erased symbols have to be simply discarded, and if they

were contributing to the minimum distance, this distance will reduce. A simple example will illustrate the point. Consider the **repetition code** in which

$$0 \rightarrow 00000$$
$$1 \rightarrow 11111$$

Here $d^* = 5$. If $r = 2$, i.e., two bits get erased (let us say the first two), we will have

$$0 \rightarrow ??000$$
$$1 \rightarrow ??111$$

Now, the effective minimum distance $d_1^* = d^* - r = 3$.

Therefore, for a channel with t errors and r erasures, $d^* - r \geq 2t + 1$. Or,

$$d^* \geq 2t + r + 1 \tag{3.9}$$

For a channel which has no errors ($t = 0$), only r erasures,

$$d^* \geq r + 1 \tag{3.10}$$

Next let us give a little more formal treatment to the decoding procedure. Can we construct some mathematical tools to simplify the nearest neighbour decoding? Suppose the codeword $\boldsymbol{c} = c_1 c_2, \ldots, c_n$ is transmitted over a noisy channel. The noise in the channel changes some or all of the symbols of the codeword. Let the received vector be denoted by $\boldsymbol{v} = v_1 v_2, \ldots, v_n$. Define the **error vector** as

$$\boldsymbol{e} = \boldsymbol{v} - \boldsymbol{c} = v_1 v_2, \ldots, v_n - c_1 c_2, \ldots, c_n = e_1 e_2, \ldots, e_n \tag{3.11}$$

The decoder has to decide from the received vector, $\boldsymbol{v}$, which codeword was transmitted, or equivalently, it must determine the error vector, $\boldsymbol{e}$.

Definition 3.17 Let $\boldsymbol{C}$ be an (n, k) code over $GF(q)$ and $\boldsymbol{a}$ be any vector of length n. Then the set

$$\boldsymbol{a} + \boldsymbol{C} = \{\boldsymbol{a} + \boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{C}\} \tag{3.12}$$

is called a **Coset** (or translate) of $\boldsymbol{C}$. $\boldsymbol{a}$ and $\boldsymbol{b}$ are said to be in the same coset if $(\boldsymbol{a} - \boldsymbol{b}) \in \boldsymbol{C}$.

Theorem 3.6 Suppose $\boldsymbol{C}$ is an (n, k) code over $GF(q)$. Then,

(i) every vector $\boldsymbol{b}$ of length n is in some coset of $\boldsymbol{C}$.

(ii) each coset contains exactly q^k vectors.

(iii) two cosets are either disjoint or coincide (partial overlap is not possible).

(iv) if $\boldsymbol{a} + \boldsymbol{C}$ is a coset of $\boldsymbol{C}$ and $\boldsymbol{b} \in \boldsymbol{a} + \boldsymbol{C}$, we have $\boldsymbol{b} + \boldsymbol{C} = \boldsymbol{a} + \boldsymbol{C}$.

Proof

(i) $\boldsymbol{b} = \boldsymbol{b} + 0 \in \boldsymbol{b} + \boldsymbol{C}$.

(ii) Observe that the mapping $\boldsymbol{C} \rightarrow \boldsymbol{a} + \boldsymbol{C}$ defined by $\boldsymbol{x} \rightarrow \boldsymbol{a} + \boldsymbol{x}$, for all $\boldsymbol{x} \in \boldsymbol{C}$ is a one-to-one mapping. Thus the cardinality of $\boldsymbol{a} + \boldsymbol{C}$ is the same as that of $\boldsymbol{C}$, which is equal to q^k.

(iii) Suppose the cosets $\boldsymbol{a} + \boldsymbol{C}$ and $\boldsymbol{a} + \boldsymbol{C}$ overlap, i.e., they have at least one vector in common.

Let $\boldsymbol{v} \in (\boldsymbol{a} + \boldsymbol{C}) \cap (\boldsymbol{b} + \boldsymbol{C})$. Thus, for some $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{C}$,

$$\boldsymbol{v} = \boldsymbol{a} + \boldsymbol{x} = \boldsymbol{b} + \boldsymbol{y}.$$

Or, $$\boldsymbol{b} = \boldsymbol{a} + \boldsymbol{x} - \boldsymbol{y} = \boldsymbol{a} + \boldsymbol{z}, \text{ where } \boldsymbol{z} \in \boldsymbol{C}$$

(because, the difference of two codewords is also a codeword).

Thus, $$\boldsymbol{b} + \boldsymbol{C} = \boldsymbol{a} + \boldsymbol{C} + \boldsymbol{z} \text{ or } (\boldsymbol{b} + \boldsymbol{C}) \subset (\boldsymbol{a} + \boldsymbol{C}).$$

Similarly, it can be shown that $(\boldsymbol{a} + \boldsymbol{C}) \subset (\boldsymbol{b} + \boldsymbol{C})$. From these two we can conclude that $(\boldsymbol{b} + \boldsymbol{C}) = (\boldsymbol{a} + \boldsymbol{C})$.

(iv) Since $\boldsymbol{b} \in \boldsymbol{a} + \boldsymbol{C}$, it implies that $\boldsymbol{b} = \boldsymbol{a} + \boldsymbol{x}$, for some $\boldsymbol{x} \in \boldsymbol{C}$.

Next, if $\boldsymbol{b} + \boldsymbol{y} \in \boldsymbol{b} + \boldsymbol{C}$, then,

$$\boldsymbol{b} + \boldsymbol{y} = (\boldsymbol{a} + \boldsymbol{x}) + \boldsymbol{y} = \boldsymbol{a} + (\boldsymbol{x} + \boldsymbol{y}) \in \boldsymbol{a} + \boldsymbol{C}.$$

Hence,

$\boldsymbol{b} + \boldsymbol{C} \subseteq \boldsymbol{a} + \boldsymbol{C}$. On the other hand, if $\boldsymbol{a} + \boldsymbol{z} \in \boldsymbol{a} + \boldsymbol{C}$, then,

$$\boldsymbol{a} + \boldsymbol{z} = (\boldsymbol{b} - \boldsymbol{x}) + \boldsymbol{z} = \boldsymbol{b} + (\boldsymbol{z} - \boldsymbol{x}) \in \boldsymbol{b} + \boldsymbol{C}.$$

Hence,

$$\boldsymbol{a} + \boldsymbol{C} \subseteq \boldsymbol{b} + \boldsymbol{C}, \text{ and so } \boldsymbol{b} + \boldsymbol{C} = \boldsymbol{a} + \boldsymbol{C}.$$

Definition 3.18 The vector having the minimum weight in a coset is called the **Coset Leader**. If there is more than one vector with the minimum weight, one of them is chosen at random and is declared the coset leader.

Example 3.17 Let $\boldsymbol{C}$ be the binary (3, 2) code with the generator matrix given by

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

i.e., $\boldsymbol{C} = \{000, 010, 101, 111\}$.

The cosets of $\boldsymbol{C}$ are,

$$000 + \boldsymbol{C} = 000, 010, 101, 111,$$
$$001 + \boldsymbol{C} = 001, 011, 100, 110.$$

Note that all the eight vectors have been covered by these two cosets. As we have already seen (in the above theorem), if $\boldsymbol{a} + \boldsymbol{C}$ is a coset of $\boldsymbol{C}$ and $\boldsymbol{b} \in \boldsymbol{a} + \boldsymbol{C}$, we have $\boldsymbol{b} + \boldsymbol{C} = \boldsymbol{a} + \boldsymbol{C}$.

Hence, all cosets have been listed. For the sake of illustration we write down the following

$$010 + \boldsymbol{C} = 010, 000, 111, 101,$$
$$011 + \boldsymbol{C} = 011, 001, 110, 101,$$
$$100 + \boldsymbol{C} = 100, 110, 001, 011,$$
$$101 + \boldsymbol{C} = 101, 111, 000, 101,$$
$$110 + \boldsymbol{C} = 110, 100, 011, 001,$$
$$111 + \boldsymbol{C} = 111, 101, 010, 000.$$

It can be seen that all these sets are already covered.

Since two cosets are either disjoint or coincide (from Theorem 3.6), the set of all vectors, $GF(q)^n$ can be written as

$$GF(q)^n = \boldsymbol{C} \cup (\boldsymbol{a}_1 + \boldsymbol{C}) \cup (\boldsymbol{a}_2 + \boldsymbol{C}) \cup \ldots \cup (\boldsymbol{a}_t + \boldsymbol{C})$$

where $$t = q^{n-k} - 1.$$

Definition 3.19 A **Standard Array** for an (n, k) code $\boldsymbol{C}$ is a $q^{n-k} \times q^k$ array of *all* vectors in $GF(q)^n$ in which the first row consists of the code $\boldsymbol{C}$ (with **0** on the extreme left), and the other rows are the cosets $\boldsymbol{a}_i + \boldsymbol{C}$, each arranged in corresponding order, with the coset leader on the left.

Steps for constructing a standard array:

(i) In the first row write down all the valid codewords, starting with the all-zero codeword.

(ii) Choose a vector $\boldsymbol{a}_1$ which is not in the first row. Write down the coset $\boldsymbol{a}_1 + \boldsymbol{C}$ as the second row such that $\boldsymbol{a}_1 + \boldsymbol{x}$ is written under $\boldsymbol{x} \in \boldsymbol{C}$.

(iii) Next choose another vector $\boldsymbol{a}_2$ (not present in the first two rows) of minimum weight and write down the coset $\boldsymbol{a}_2 + \boldsymbol{C}$ as the third row such that $\boldsymbol{a}_2 + \boldsymbol{x}$ is written under $\boldsymbol{x} \in \boldsymbol{C}$.

(iv) Continue the process until all the cosets are listed and every vector in $GF(q)^n$ appears exactly once.

Example 3.18 Consider the code $\boldsymbol{C} = \{0000, 1011, 0101, 1110\}$. The corresponding standard array is

codewords →	0000	1011	0101	1110
	1000	0011	1101	0110
	0100	1111	0001	1010
	0010	1001	0111	1100
	↑			
	coset leader			

Note that each entry is the sum of the codeword and its coset leader.

Let us now look at the concept of decoding (obtaining the information symbols from the received codewords) using the standard array. Since the standard array comprises all possible words belonging to $GF(q)^n$, the received word can always be identified with one of the elements of the standard array. If the received word is a valid codeword, it is concluded that no errors have occurred (this conclusion may be wrong with a very low probability of error, when one valid codeword gets modified to another valid codeword due to noise!). In the case that the received word, $\boldsymbol{v}$, does not belong to the set of valid codewords, we surmise that an error has occurred. The decoder then declares that the coset leader is the error vector, $\boldsymbol{e}$, and decodes the codeword as $\boldsymbol{v} - \boldsymbol{e}$. This is the codeword at the top of the column containing $\boldsymbol{v}$. Thus, mechanically, we decode the codeword as the one on the top of the column containing the received word.

Example 3.19 Suppose the code in the previous example $C = \{0000, 1011, 0101, 1110\}$ is used and the received word is $v = 1101$. Since it is not one of the valid codewords, we deduce that an error has occurred. Next we try to estimate which one of the four possible codewords was actually transmitted. If we make use of the standard array of the earlier example, we find that 1101 lies in the 3rd column. The topmost entry of this column is 0101. Hence the estimated codeword is 0101. Observe that:

$$d\,(1101, 0000) = 3,\ d\,(1101, 1011) = 2,$$
$$d\,(1101, 0101) = 1,\ d\,(1101, 1110) = 2$$

and the error vector $e = 1000$, the coset leader.

Codes with larger blocklengths are desirable (though not always; see the concluding remarks on this chapter) because the code rates of larger codes perform closer to the **Shannon Limit.** As we go to larger codes (with larger values of k and n), the method of standard array will become less practical because the size of the standard array $(q^{n-k} \times q^k)$ will become unmanageably large. One of the basic objectives of coding theory is to develop efficient decoding strategies. If we are to build decoders that will work in real-time, the decoding scheme should be realizable both in terms of memory required as well as the computational load. Is it possible to reduce the standard array? The answer lies in the concept of Syndrome Decoding, which we are going to discuss next.

3.7 SYNDROME DECODING

The standard array can be simplified if we store only the first column, and compute the remaining columns, if needed. To do so, we introduce the concept of the ***Syndrome*** of the error pattern.

Definition 3.20 Suppose H is a parity check matrix of an (n, k) code, then for any vector $v \in GF(q)^n$, the vector

$$s = vH^T \tag{3.13}$$

is called the **Syndrome** of v.

The syndrome of v is sometimes explicitly written as $s(v)$. It is called a syndrome because it gives us the symptoms of the error, thereby helping us to diagnose the error.

Theorem 3.7 Two vectors x and y are in the same coset of C if and only if they have the same syndrome.

Proof The vectors x and y belong to the same coset

$$\Leftrightarrow\ x + C = y + C$$
$$\Leftrightarrow\ x - y \in C$$
$$\Leftrightarrow\ (x - y)H^T = 0$$
$$\Leftrightarrow\ xH^T = yH^T$$
$$\Leftrightarrow\ s(x) = s(y)$$

Thus, there is a one to one correspondence between cosets and syndromes.

We can reduce the size of the standard array by simply listing the syndromes and the corresponding coset leaders.

Example 3.20 **We now extend the standard array listed in Example 3.18 by adding the syndrome column.**

The code is $C = \{0000, 1011, 0101, 1110\}$. The corresponding standard array is

					Syndrome
Codewords →	0000	1011	0101	1111	00
	1000	0011	1101	0110	11
	0100	1111	0001	1010	01
	0010	1001	0111	1100	10
	↑				
	coset leader				

The steps for syndrome decoding are as follows

(i) Determine the syndrome $(s = vH^T)$ of the received word, v.
(ii) Locate the syndrome in the 'syndrome column'.
(iii) Determine the corresponding coset leader. This is the error vector, e.
(iv) Subtract this error vector from the received word to get the codeword $y = v - e$.

Having developed an efficient decoding methodology by means of syndrome decoding, let us now find out how much advantage coding actually provides.

3.8 ERROR PROBABILITY AFTER CODING (PROBABILITY OF ERROR CORRECTION)

Definition 3.21 The Probability of Error (or, the Word Error Rate) P_{er} for any decoding scheme is the probability that the decoder output is a wrong codeword. It is also called the Residual Error Rate.

Suppose there are M codewords (of length n) which are used with equal probability. Let the decoding be done using a standard array. Let the number of coset leaders with weight i be denoted by α_i. We assume that the channel is a BSC with symbol error probability p. A decoding error occurs if the error vector e is *not* a coset leader. Therefore, the probability of correct decoding will be

$$P_{cor} = \sum_{i=0}^{n} \alpha_i \, p^i \, (1-p)^{n-i} \tag{3.14}$$

Hence, the probability of error will be

$$P_{err} = 1 - \sum_{i=0}^{n} \alpha_i \, p^i \, (1-p)^{n-i} \tag{3.15}$$

Example 3.21 Consider the standard array in Example 3.18. The coset leaders are 0000, 1000, 0100 and 0010. Therefore $\alpha_0 = 1$ (only one coset leader with weight equal to zero), $\alpha_1 = 3$ (the remaining three are of weight one) and all other $\alpha_i = 0$.

Therefore,

$$P_{err} = 1 - [(1-p)^4 + 3p(1-p)^3]$$

Recall that this code has four codewords, and can be used to send 2 bits at a time. If we did not perform coding, the probability of error of the 2-bit message being received incorrectly would be

$$P_{err} = 1 - P_{cor} = 1 - (1-p)^2.$$

Note that for $p = 0.01$, the **Word Error Rate** (upon coding) is $P_{err} = 0.0103$, while for the uncoded case $P_{err} = 0.0199$. So, coding has almost halved the word error rate. The comparison of P_{err} for messages with and without coding is plotted in Fig. 3.3. It can be seen that coding outperforms the uncoded case only for $p < 0.5$. Note that the improvement due to coding comes at the cost of information transfer rate. In this example, the rate of information transfer has been cut down by half as we are sending two parity bits for every two information bits.

Fig. 3.3 *Comparison of P_{err} for Coded and Uncoded 2-Bit Messages.*

Example 3.22 This example will help us visualize the power of coding. Consider a BSC with the probability of symbol error $p = 10^{-7}$. Suppose 10 bit long words are being transmitted *without coding*. Let the bit rate of the transmitter be 10^7 b/s, which implies that 10^6 word/s are being sent. The probability that a word is received incorrectly is

$$\binom{10}{1}(1-p)^9 p + \binom{10}{2}(1-p)^8 p^2 + \binom{10}{3}(1-p)^7 p^3 + \ldots \approx \binom{10}{1}(1-p)^9 p = 10^{-6} \text{ word/s.}$$

Therefore, in one second, $10^{-6} \times 10^6 = 1$ word will be in error ! The implication is that every second a word will be in error and *it will not be detected.*

Next, let us add a parity bit to the uncoded words so as to make them 11 bits long. The parity makes all the codewords of even parity and thus ensures that a single bit in error will be detected. The only way that the coded word will be in error is if two or more bits get flipped, i.e., at least two bits are in error. This can be computed as 1 – probability that less than two bits are in error. Therefore, the probability of word error will be

$$1-(1-p)^{11} - \binom{11}{1}(1-p)^{10} p \approx 1-(1-11p) - 11(1-10p)\,p = 110\,p^2 = 11 \times 10^{-13}$$

The new word rate will be $10^7/11$ word/s because now 11 bits constitute one word and the bit rate is the same as before. Thus in one second, $(10^7/11) \times (11 \times 10^{-13}) = 10^{-6}$ words will be in error. This implies that, after coding, one word will be received incorrectly *without detection* every 10^6 seconds = 11.5 days!

So just by increasing the word length from 10 bits (uncoded) to 11 bits (with coding), we have been able to obtain a dramatic decrease in the Word Error Rate. For the second case, each time a word is detected to be in error, we can request the transmitter to re-transmit the word.

This strategy for retransmission is called the **A**utomatic **R**epeat **R**equest (ARQ).

3.9 PERFECT CODES

Definition 3.22 For any vector $\boldsymbol{u}$ in $GF(q)^n$ and any integer $r \geq 0$, the sphere of radius r and centre $\boldsymbol{u}$, denoted by $S(\boldsymbol{u}, r)$, is the set $\{\mathbf{v} \in GF(q)^n \mid d(\boldsymbol{u}, \boldsymbol{v}) \leq r\}$.

This definition can be interpreted graphically, as shown in Fig. 3.4. Consider a code $\boldsymbol{C}$ with minimum distance $d^*(\boldsymbol{C}) \geq 2t+1$. The spheres of radius t centred at the codewords $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots \boldsymbol{c}_M\}$ of $\boldsymbol{C}$ will then be disjoint. Now consider the decoding problem. Any received vector can be represented as a point in this space. If this point lies within a sphere, then by nearest neighbour decoding it will be decoded as the centre of the sphere. If t or fewer errors occur, the received word will definitely lie within the sphere of the codeword that was transmitted, and will be correctly decoded. If, however, larger than t errors occur, it will escape the sphere, thus resulting in incorrect decoding.

Fig. 3.4 *The concept of spheres in $GF(q)^n$.*

The codewords of the code with $d^*(C) \geq 2t + 1$ are the centres of these non-overlapping spheres.

Theorem 3.8 A sphere of radius r $(0 \leq r \leq n)$ contains exactly

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{r}(q-1)^r \text{ vectors.} \quad (3.16)$$

Proof Consider a vector $\boldsymbol{u}$ in $GF(q)^n$ and another vector $\boldsymbol{v}$ which is at a distance m from $\boldsymbol{u}$. This implies that the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ differ at exactly m places. The total number of ways in which m position can be chosen from n positions is $\binom{n}{m}$. Now, each of these m places can be replaced by $(q-1)$ possible symbols. This is because the total size of the alphabet is q, out of which one is currently being used in that particular position in $\boldsymbol{u}$. Hence, the number of vectors at a distance exactly m from $\boldsymbol{u}$ is

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{r}(q-1)^r \quad (3.17)$$

Example 3.23 **Consider a binary code (i.e., $q = 2$) and blocklength $n = 4$. The number of vectors at a distance 2 or less from any codeword will be**

$$\binom{4}{0} + \binom{4}{1}(1) + \binom{4}{2}(1)^2 = 1 + 4 + 6 = 11$$

Without loss of generality we can choose the fixed vector $\boldsymbol{u} = 0000$. The vectors of distance 2 or less are

vectors at a distance 2: 0011, 1001, 1010, 1100, 0110, 0101,
vectors at a distance 1: 0001, 0010, 0100, 1000,
vectors at a distance 0: 0000.
Thus, there is a total of 11 such vectors.

Theorem 3.9 A q-ary (n, k) code with M codewords and minimum distance $(2t + 1)$ satisfies

$$M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right\} \leq q^n \quad (3.18)$$

Proof Suppose C is a q-ary (n, k) code. Consider spheres of radius t centred on the M codewords. Each sphere of radius t has

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t$$

vectors (theorem 3.8). Since none of the spheres intersect, the total number of vectors for the M disjoint spheres is $M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right\}$ which is upper bounded by q^n, the total number of vectors of length n in $GF(q)^n$.

This bound is called the **Hamming Bound** or the **Sphere Packing Bound** and it holds good for nonlinear codes as well. For binary codes, the **Hamming Bound** will become

$$M\left\{\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}\right\} \leq 2^n. \quad (3.19)$$

It should be noted here that the mere existence of a set of integers n, M and t satisfying the Hamming bound, does not confirm it as a binary code. For example, the set $n = 5$, $M = 5$ and $t = 1$ satisfies the Hamming Bound. However, no binary code exists for this specification.

Observe that for the case when $M = q^k$, the Hamming Bound may be alternatively written as

$$\log_q\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right\} \leq n - k. \quad (3.20)$$

Definition 3.23 A Perfect Code is one which achieves the Hamming Bound, i.e.,

$$M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right\} = q^n. \quad (3.21)$$

For a Perfect Code, there are equal radius disjoint spheres centred at the codewords which completely fill the space. Thus, a t error correcting perfect code utilizes the entire space in the most efficient manner.

Example 3.24 Consider the Binary Repetition Code

$$C = \begin{cases} 00\ldots0 \\ 11\ldots1 \end{cases}$$

of block length n, where n is odd. In this case $M = 2$ and $t = (n-1)/2$. Upon substituting these values in the left hand side of the inequality for Hamming bound we get

$$\text{LHS} = \left\{\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{(n-1)/2}\right\} = 2.2^{n-1} = 2^n = \text{RHS}$$

Thus the repetition code is a Perfect Code. It is actually called a *Trivial* Perfect Code. In the next chapter, we shall see some examples of Non-trivial Perfect Codes.

One of the ways to search for perfect codes is to obtain the integer solutions for the parameters n, q, M and t in the equation for Hamming bound. Some of the solutions found by exhaustive computer search are listed below.

S.No.	n	q	M	t
1	23	2	2^{12}	3
2	90	2	2^{78}	2
3	11	3	3^6	2

3.10 HAMMING CODES

There are both binary and non-binary Hamming Codes. Here, we shall limit our discussion to binary Hamming Codes. The binary Hamming Codes have the property that

$$(n, k) = (2^m - 1,\quad 2^m - 1 - m) \tag{3.22}$$

where m is any positive integer. For example, for $m = 3$ we have a (7, 4) Hamming Code. The parity check matrix, $\boldsymbol{H}$, of a Hamming Code is a very interesting matrix. Recall that the parity check matrix of an (n, k) code has $n - k$ rows and n columns. For the binary (n, k) Hamming code, the $n = 2^m - 1$ columns consist of all possible binary vectors with $n - k = m$ elements, except the all zero vector.

Example 3.25 The generator matrix for the binary (7, 4) Hamming Code is given by

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The corresponding parity check matrix is

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Observe that the columns of the parity check matrix consist of (100), (010), (101), (110), (111), (011) and (001). These seven are all the possible non-zero binary vectors of length three. It is quite easy to generate a systematic Hamming Code. The parity check matrix $\boldsymbol{H}$ can be arranged in the systematic form as follows

$$\boldsymbol{H} = \left[\begin{array}{ccc:ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = [-\boldsymbol{P}^T \mid \boldsymbol{I}].$$

Thus, the generator matrix in the systematic form for the binary Hamming code is

$$\boldsymbol{G} = [\boldsymbol{I} \mid \boldsymbol{P}] = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

From the above example, we observe that no two columns of $\boldsymbol{H}$ are linearly dependent (otherwise they would be identical). However, for $m > 1$, it is possible to identify three columns of $\boldsymbol{H}$ that would add up to zero. Thus, the minimum distance, d^*, of an (n, k) Hamming Code is equal to 3, which implies that it is a single-error correcting code. Hamming Codes are Perfect Codes.

By adding an overall parity bit, an (n, k) Hamming Code can be modified to yield an $(n+1, k)$ code with $d^* = 4$. On the other hand, an (n, k) Hamming Code can be shortened to an $(n - l, k - l)$ code by removing l rows of its generator matrix $\boldsymbol{G}$ or, equivalently, by removing l columns of its parity check matrix, $\boldsymbol{H}$. We can now give a more formal definition of Hamming Codes.

Definition 3.24 Let $n = (q^k - 1)/(q - 1)$. The (n, k) **Hamming Code over** $GF(q)$ is a code for which the parity check matrix has columns that are pairwise linearly independent (over $GF(q)$), i.e., the columns are a maximal set of pairwise linearly independent vectors.

3.11 OPTIMAL LINEAR CODES

Definition 3.25 For an (n, k, d^*) **Optimal Code**, no $(n-1, k, d^*)$, $(n+1, k+1, d^*)$ or $(n+1, k, d^*+1)$ code exists.

Optimal Linear Codes give the best distance property under the constraint of the block length. Most of the optimal codes have been found by long computer searches. It may be possible to have more than one optimal code for a given set of parameters n, k and d^*. For instance, there exist two different binary (25, 5, 10) optimal codes.

Example 3.26 **Consider the (24, 12, 8) binary code. It can be verified that**

(i) a (23, 12, 8) code does not exist (only the (23, 12, 7) code exists),

(ii) a (25, 13, 8) code does not exist,

(iii) a (25, 12, 9) code does not exist.

Thus the binary (24, 12, 8) code is an optimal code.

3.12 MAXIMUM DISTANCE SEPARABLE (MDS) CODES

In this section we consider the problem of finding as large a minimum distance d^* possible for a given redundancy, r.

Theorem 3.10 An $(n, n-r, d^*)$ code satisfies $d^* \le r + 1$.

Proof From the Singleton Bound we have $d^* \le n - k + 1$.

Substitute $k = n - r$ to get $\quad d^* \le r + 1$.

Definition 3.26 An $(n, n-r, r+1)$ **code is called a Maximum Distance Separable (MDS) code. An MDS code is a linear code of redundancy** r**, whose minimum distance is equal to** $r + 1$.

3.13 CONCLUDING REMARKS

The classic paper by Claude Elwood Shannon in the *Bell System Technical Journal* in 1948 gave birth to two important fields (i) Information Theory and (ii) Coding Theory. At that time, Shannon was only 32 years old. According to Shannon's Channel Coding Theorem, *"the error rate of data transmitted over a band-limited noisy channel can be reduced to an arbitrarily small amount if the information rate is less than the channel capacity"*. Shannon predicted the existence of good channel codes but did not construct them. Since then the search for good codes has been on. Shannon's seminal paper can be accessed from the site:

http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

In 1950, R.W. Hamming introduced the first single-error correcting code, which is still used today. The work on linear codes was extended by Golay (whose codes will be studied in the following chapter). Golay also introduced the concept of Perfect Codes. Non-binary Hamming Codes were developed by Golay and Cocke in the late 1950s. Lately, a lot of computer searches have been used to find interesting codes. However, some of the best known codes are ones discovered by sheer genius rather than exhaustive searches.

According to Shannon's theorem, if $C(p)$ represents the capacity (see Chapter 1 for further details) of a BSC with probability of bit error equal to p, then for arbitrarily low probability of symbol error we must have the code rate $R < C(p)$. Even though the channel capacity provides an upperbound on the achievable code rate ($R = k/n$), evaluating a code exclusively against channel capacity may be misleading. The block length of the code, which translates directly into delay, is also an important parameter. Even if a code performs far from ideal, it is possible that it is the best possible code for a given rate and length. It has been observed that as we increase the block length of codes, the bounds on code rate are closer to channel capacity as opposed to codes with smaller blocklengths. However, longer blocklengths imply longer delays in decoding. This is because decoding of a codeword cannot begin until we have received the entire codeword. The maximum delay allowable is limited by practical constraints. For example, in mobile radio communications, packets of data are restricted to fewer than 200 bits. In these cases, codewords with very large blocklengths cannot be used.

SUMMARY

- A Word is a sequence of symbols. A Code is a set of vectors called codewords.
- The Hamming Weight of a codeword (or any vector) is equal to the number of non-zero elements in the codeword. The Hamming Weight of a codeword c is denoted by $w(c)$.
- A Block Code consists of a set of fixed length codewords. The fixed length of these codewords is called the Block Length and is typically denoted by n. A Block Coding Scheme converts a block of k information symbols to n coded symbols. Such a code is denoted by (n, k).
- The Code Rate of an (n, k) code is defined as the ratio (k/n), and reflects the fraction of the codeword that consists of the information symbols.
- The minimum distance of a code is the minimum Hamming Distance between any two codewords. An (n, k) code with minimum distance d^* is sometimes denoted by (n, k, d^*). The minimum weight of a code is the smallest weight of any non-zero codeword, and is

denoted by w^*. For a Linear Code the minimum distance is equal to the minimum weight of the code, i.e., $d^* = w^*$.

- A Linear Code has the following properties:
  - (i) The sum of two codewords belonging to the code is also a codeword belonging to the code.
  - (ii) The all-zero codeword is always a codeword.
  - (iii) The minimum Hamming Distance between two codewords of a linear code is equal to the minimum weight of any non-zero codeword, i.e., $d^* = w^*$.
- The generator matrix converts (encodes) a vector of length k to a vector of length n. Let the input vector (uncoded symbols) be represented by $\boldsymbol{i}$. The coded symbols will be given by $\boldsymbol{c} = \boldsymbol{iG}$.
- Two q-ary codes are called equivalent if one can be obtained from the other by one or both operations listed below:
  - (i) permutation of symbols appearing in a fixed position.
  - (ii) permutation of position of the code.
- An (n, k) Systematic Code is one in which the first k symbols of the codeword of block length n are the information symbols themselves. A generator matrix of the form $\boldsymbol{G} = [\boldsymbol{I}\,|\boldsymbol{P}]$ is called the systematic form or the standard form of the generator matrix, where $\boldsymbol{I}$ is a $k \times k$ identity matrix and $\boldsymbol{P}$ is a $k \times (n-k)$ matrix.
- The Parity Check Matrix, $\boldsymbol{H}$, for the given code satisfies $\boldsymbol{cH}^T = \boldsymbol{0}$, where $\boldsymbol{c}$ is a valid codeword. Since $\boldsymbol{c} = \boldsymbol{iG}$, therefore, $\boldsymbol{iGH}^T = \boldsymbol{0}$. The Parity Check Matrix is not unique for a given code.
- A Maximum Distance Code satisfies $d^* = n - k + 1$.
- For a code to be able to correct up to t errors, we must have $d^* \geq 2t + 1$, where d^* is minimum distance of the code.
- Let $\boldsymbol{C}$ be an (n, k) code over $GF(q)$ and $\boldsymbol{a}$ be any vector of length n. Then the set $\boldsymbol{a} + \boldsymbol{C} = \{\boldsymbol{a} + \boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{C}\}$ is called a coset (or translate) of $\boldsymbol{C}$. $\boldsymbol{a}$ and $\boldsymbol{b}$ are said to be in the same coset iff $(\boldsymbol{a} - \boldsymbol{b}) \in \boldsymbol{C}$.
- Suppose $\boldsymbol{H}$ is a Parity Check Matrix of an (n, k) code. Then for any vector $\boldsymbol{v} \in GF(q)^n$, the vector $\boldsymbol{s} = \boldsymbol{vH}^T$ is called the Syndrome of $\boldsymbol{v}$. It is called a syndrome because it gives us the symptoms of the error, thereby helping us to diagnose the error.
- A Perfect Code achieves the Hamming Bound, i.e.,

$$M = \left\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\right\} = q^n$$

- The binary Hamming Codes have the property that $(n, k) = (2^m - 1, 2^m - 1 - m)$, where m is any positive integer. Hamming Codes are Perfect Codes.
- For an (n, k, d^*) Optimal Code, no $(n-1, k, d^*)$, $(n+1, k+1, d^*)$ or $(n+1, k, d^*+1)$ code exists.
- An $(n, n-r, r+1)$ code is called a Maximum Distance Separable (MDS) Code. An MDS code is a linear code of redundancy r, whose minimum distance is equal to $r + 1$.

The greatest unsolved theorem in mathematics is why some people are better at it than others.

Mathesis, Adrian

PROBLEMS

3.1 Show that $\boldsymbol{C} = \{0000, 1100, 0011, 1111\}$ is a linear code. What is its minimum distance?

3.2 Construct, if possible, binary (n, k, d^*) codes with the following parameters:
- (i) (6, 1, 6)
- (ii) (3, 3, 1)
- (iii) (4, 3, 2)

3.3 Consider the following generator matrix over $GF(2)$

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

- (i) Generate all possible codewords using this matrix.
- (ii) Find the parity check matrix, $\boldsymbol{H}$.
- (iii) Find the generator matrix of an equivalent systematic code.
- (iv) Construct the standard array for this code.
- (v) What is the minimum distance of this code?
- (vi) How many errors can this code detect?
- (vii) Write down the set of error patterns this code can detect.
- (viii) How many errors can this code correct?
- (ix) What is the probability of symbol error if we use this encoding scheme? Compare it with the uncoded probability of error.
- (x) Is this a linear code?

3.4 For the code $\boldsymbol{C} = \{00000, 10101, 01010, 11111\}$ construct the generator matrix. Since this $\boldsymbol{G}$ is not unique, suggest another generator matrix that can also generate this set of codewords.

3.5 Show that if there is a binary (n, k, d) code with d^* even, then there exists a binary (n, k, d^*) code in which all codewords have even weight.

3.6 Show that if $\boldsymbol{C}$ is a binary linear code, then the code obtained by adding an overall parity check bit to $\boldsymbol{C}$ is also linear.

3.7 For each of the following sets S, list the code $<S>$.
- (a) $\boldsymbol{S} = \{0101, 1010, 1100\}$.
- (b) $\boldsymbol{S} = \{1000, 0100, 0010, 0001\}$.
- (c) $\boldsymbol{S} = \{11000, 01111, 11110, 01010\}$.

3.8 Consider the (23, 12, 7) binary code. Show that if it is used over a binary symmetric channel (BSC) with probability of bit error $p = 0.01$, the word error will be approximately 0.00008.

3.9 Suppose $\boldsymbol{C}$ is a binary code with parity check matrix, $\boldsymbol{H}$. Show that the extended code $\boldsymbol{C}_1$, obtained from $\boldsymbol{C}$ by adding an overall parity bit, has the parity check matrix

$$\boldsymbol{H}_1 = \left[\begin{array}{cccccc} & & & & | & 0 \\ & & & & & 0 \\ & H & & & | & \\ & & & & | & 0 \\ - & - & - & - & - & - \\ 1 & 1 & \cdots & 1 & | & 1 \end{array}\right]$$

3.10 For a (5, 3) code over $GF(4)$, the generator matrix is given by

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 3 \end{bmatrix}$$

(i) Find the parity check matrix.
(ii) How many errors can this code detect?
(iii) How many errors can this code correct?
(iv) How many erasures can this code correct?
(v) Is this a perfect code?

3.11 Let $\boldsymbol{C}$ be a binary perfect code of length n with minimum distance 7. Show that $n = 7$ or $n = 23$.

3.12 Let r_H denote the code rate for the binary Hamming code. Determine $\lim_{k \to \infty} r_H$.

3.13 Show that a (15, 8, 5) code does not exist.

COMPUTER PROBLEMS

3.14 Write a computer program to find the minimum distance of a Linear Block Code over $GF(2)$, given the generator matrix for the code.

3.15 Generalize the above program to find the minimum distance of any Linear Block Code over $GF(q)$.

3.16 Write a computer program to exhaustively search for all the perfect code parameters n, q, M and t in the equation for the Hamming Bound. Search for $1 \geq n \geq 200$, $2 \geq q \geq 11$.

3.17 Write a computer program for a universal binary Hamming encoder with rate $\dfrac{2^m - 1}{2^m - 1 - m}$.

The program should take as input the value of m and a bit-stream to be encoded. It should then generate an encoded bit-stream. Develop a program for the decoder also.

Now, perform the following tasks:

(i) Write an error generator module that takes in a bit stream and outputs another bit-stream after inverting every bit with probability p, i.e., the probability of a bit error is p.
(ii) For $m = 3$, pass the Hamming encoded bit-stream through the above-mentioned module and then decode the received words using the decoder block.
(iii) Plot the residual error probability (the probability of error after decoding) as a function of p. Note that if you are working in the range of BER $= 10^{-r}$, you must transmit of the order of 10^{r+2} bits (why?).
(iv) Repeat your simulations for $m = 5$, 8 and 15. What happens as $m \to \infty$.

4

Cyclic Codes

We arrive at truth, not by reason only, but also by the heart.

Pascal, Blaise (1623-1662)

4.1 INTRODUCTION TO CYCLIC CODES

In the previous chapter, while dealing with Linear Block Codes, certain linearity constraints were imposed on the structure of the block codes. These structural properties help us to search for good linear block codes that are fast and easy to encode and decode. In this chapter, we shall explore a *subclass* of linear block codes which has another constraint on the structure of the codes. The additional constraint is that any cyclic shift of a codeword results in another valid codeword. This condition allows very simple implementation of these cyclic codes by using shift registers. Efficient circuit implementation is a selling feature of any error control code. We shall also see that the theory of Galois Field can be used effectively to study, analyze and discover new cyclic codes. The Galois Field representation of cyclic codes leads to low-complexity encoding and decoding algorithms.

This chapter is organized as follows. In the first two sections, we take a mathematical detour to polynomials. We will review some old concepts and learn a few new ones. Then, we will use these mathematical tools to construct and analyze cyclic codes. The matrix description of cyclic

codes will be introduced next. We will then, discuss some popular cyclic codes. The chapter will conclude with a discussion on circuit implementation of cyclic codes.

Definition 4.1 A code C is **cyclic** if

(i) C is a linear code, and,

(ii) any cyclic shift of a codeword is also a codeword, i.e., if the codeword $a_0a_1...a_{n-1}$ is in C then $a_{n-1}a_0...a_{n-2}$ is also in C.

Example 4.1 The binary code C_1 = {0000, 0101, 1010, 1111} is a cyclic code. However C_2 = {0000, 0110, 1001, 1111} is not a cyclic code, but is *equivalent* to the first code. Interchanging the third and the fourth components of C_2 yields C_1.

4.2 POLYNOMIALS

Definition 4.2 A **polynomial** is a mathematical expression

$$f(x) = f_0 + f_1x + ... + f_mx^m, \tag{4.1}$$

where the symbol x is called the indeterminate and the coefficients $f_0, f_1, ..., f_m$ are the elements of $GF(q)$. The coefficient f_m is called the leading coefficient. If $f_m \neq 0$, then m is called the **degree** of the polynomial, and is denoted by deg $f(x)$.

Definition 4.3 A polynomial is called **monic** if its leading coefficient is unity.

Example 4.2 $f(x) = 3 + 7x + x^3 + 5x^4 + x^6$ is a monic polynomial over $GF(8)$. The degree of this polynomial is 6.

Polynomials play an important role in the study of cyclic codes, the subject of this chapter. Let $F[x]$ be the set of polynomials in x with coefficients in $GF(q)$. Different polynomials in $F[x]$ can be added, subtracted and multiplied in the usual manner. $F[x]$ is an example of an algebraic structure called a *ring*. A ring satisfies the first seven of the eight axioms that define a field (see Sec. 3.2 of Chapter 3). $F[x]$ is not a field because polynomials of degree greater than zero do not have a multiplicative inverse. It can be seen that if $f(x), g(x) \in F[x]$, then deg $(f(x)g(x))$ = deg $f(x)$ + deg $g(x)$. However, deg $(f(x) + g(x))$ is not necessarily max{ deg $f(x)$, deg $g(x)$}. For example, consider the two polynomials, $f(x)$ and $g(x)$, over $GF(2)$ such that $f(x) = 1 + x^2$ and $g(x) = 1 + x + x^2$. Then, deg $(f(x) + g(x))$ = deg $(x) = 1$. This is because, in $GF(2)$, $1 + 1 = 0$, and $x^2 + x^2 = (1 + 1)\, x^2 = 0$.

Example 4.3 Consider the polynomials $f(x) = 2 + x + x^2 + 2x^4$ and $g(x) = 1 + 2x^2 + 2x^4 + x^5$ over $GF(3)$. Then,

$$f(x) + g(x) = (2 + 1) + x + (1 + 2)x^2 + (2 + 2)x^4 + x^5 = x + x^4 + x^5.$$

$$\begin{aligned} f(x).\ g(x) &= (2 + x + x^2 + 2x^4)(1 + 2x^2 + 2x^4 + x^5) \\ &= 2 + x + (1 + 2.2)x^2 + 2x^3 + (2 + 2 + 2.2)x^4 + (2 + 2)x^5 \\ &\quad + (1 + 2 + 1)x^6 + x^7 + 2.2x^8 + 2x^9 \\ &= 2 + x + (1 + 1)x^2 + 2x^3 + (2 + 2 + 1)x^4 + (2 + 2)x^5 + (1 + 2 + 1)x^6 + x^7 + x^8 + 2x^9 \\ &= 2 + x + 2x^2 + 2x^3 + 2x^4 + x^5 + x^6 + x^7 + x^8 + 2x^9 \end{aligned}$$

Note that the addition and multiplication of the coefficients have been carried out in $GF(3)$.

Example 4.4 Consider the polynomial $f(x) = 1 + x$ over $GF(2)$.

$$(f(x))^2 = 1 + (1 + 1)x + x^2 = 1 + x^2$$

Again consider $f(x) = 1 + x$ over $GF(3)$.

$$(f(x))^2 = 1 + (1+1)x + x^2 = 1 + 2x + x^2$$

4.3 THE DIVISION ALGORITHM FOR POLYNOMIALS

The Division Algorithm states that, for every pair of polynomial $a(x)$ and $b(x) \neq 0$ in $F[x]$, there exists a unique pair of polynomials $q(x)$, the quotient, and $r(x)$, the remainder, such that $a(x) = q(x)\ b(x) + r(x)$, where deg $r(x) <$ deg $b(x)$. The remainder is sometimes also called the **residue**, and is denoted by $R_{b(x)}[a(x)] = r(x)$.

Two important properties of residues are

(i) $R_{f(x)}[a(x) + b(x)] = R_{f(x)}[a(x)] + R_{f(x)}[b(x)]$, and (4.2)

(ii) $R_{f(x)}[a(x).\ b(x)] = R_{f(x)}\{R_{f(x)}[a(x)].\ R_{f(x)}[b(x)]\}$ (4.3)

where $a(x)$, $b(x)$ and $f(x)$ are polynomials over $GF(q)$.

Example 4.5 Let the polynomials, $a(x) = x^3 + x + 1$ and $b(x) = x^2 + x + 1$ be defined over $GF(2)$. We can carry out the long division of $a(x)$ by $b(x)$ as follows

$$\begin{array}{rl} & \quad x + 1 \longleftarrow q(x) \\ b(x) \longrightarrow x^2 + x + 1 \,\big) & x^3 + \quad x + 1 \longleftarrow a(x) \\ & x^3 + x^2 + x \\ \hline & \quad x^2 + \quad 1 \\ & \quad x^2 + x + 1 \\ \hline & \qquad x \longleftarrow r(x) \end{array}$$

Thus, $a(x) = (x+1)\ b(x) + x$. Hence, we may write $a(x) = q(x)\ b(x) + r(x)$, where $q(x) = x + 1$ and $r(x) = x$. Note that deg $r(x) <$ deg $b(x)$.

> **Definition 4.4** Let $f(x)$ be a fixed polynomial in $F[x]$. Two polynomials, $g(x)$ and $h(x)$ in $F[x]$ are said to be **congruent modulo** $f(x)$, depicted by $g(x) \equiv h(x) \pmod{f(x)}$, if $g(x) - h(x)$ is divisible by $f(x)$.

Example 4.6 Let the polynomials $g(x) = x^9 + x^2 + 1$, $h(x) = x^5 + x^2 + 1$ and $f(x) = x^4 + 1$ be defined over $GF(2)$. Since $g(x) - h(x) = x^5 f(x)$, we can write $g(x) \equiv h(x) \pmod{f(x)}$.

Next, let us denote $F[x]/f(x)$ as the *set of polynomials* in $F[x]$ of degree less than deg $f(x)$, with addition and multiplication carried out modulo $f(x)$ as follows:

(i) If $a(x)$ and $b(x)$ belong to $F[x]/f(x)$, then the sum $a(x) + b(x)$ in $F[x]/f(x)$ is the same as in $F[x]$. This is because deg $a(x) <$ deg $f(x)$, deg $b(x) <$ deg $f(x)$ and therefore deg $(a(x) + b(x)) <$ deg $f(x)$.

(ii) The product $a(x)b(x)$ is the unique polynomial of degree less than deg $f(x)$ to which $a(x)b(x)$ (multiplication being carried out in $F[x]$) is congruent modulo $f(x)$.

$F[x]/f(x)$ is called the *ring of polynomials* (over $F[x]$) modulo $f(x)$. As mentioned earlier, a ring satisfies the first seven of the eight axioms that define a field. A ring in which every element also has a multiplicative inverse forms a field.

Example 4.7 Consider the product $(x + 1)^2$ in $F[x]/(x^2 + x + 1)$ defined over $GF(2)$. $(x + 1)^2 = x^2 + x + x + 1 = x^2 + 1 = x \pmod{x^2 + x + 1}$.

The product $(x + 1)^2$ in $F[x]/(x^2 + 1)$ defined over $GF(2)$ can be expressed as $(x + 1)^2 = x^2 + x + x + 1 = x^2 + 1 = 0 \pmod{x^2 + x + 1}$.

The product $(x + 1)^2$ in $F[x]/(x^2 + x + 1)$ defined over $GF(3)$ can be expressed as $(x + 1)^2 = x^2 + x + x + 1 = x^2 + 2x + 1 = x \pmod{x^2 + x + 1}$.

If $f(x)$ has degree n, then the ring $F[x]/f(x)$ over $GF(q)$ consists of polynomials of degree $\leq n - 1$. The size of ring will be q^n because each of the n coefficients of the polynomials can be one of the q elements in $GF(q)$.

Example 4.8 Consider the ring $F[x]/(x^2 + x + 1)$ defined over $GF(2)$. This ring will have polynomials with highest degree = 1. This ring contains $q^n = 2^{22} = 4$ elements (each element is a polynomial). The elements of the ring will be 0, 1, x and $x + 1$. The addition and multiplication tables can be written as follows.

+	0	1	x	x+1
0	0	1	x	x+1
1	1	0	x+1	x
x	x	x+1	0	1
x+1	x+1	x	1	0

.	0	1	x	x+1
0	0	0	0	0
1	0	1	x	x+1
x	0	x	x+1	1
x+1	0	x+1	1	x

Next, consider $F[x]/(x^2 + 1)$ defined over $GF(2)$. The elements of the ring will be 0, 1, x and $x + 1$. The addition and multiplication tables can be written as follows.

+	0	1	x	x+1
0	0	1	x	x+1
1	1	0	x+1	x
x	x	x+1	0	1
x+1	x+1	x	1	0

.	0	1	x	x+1
0	0	0	0	0
1	0	1	x	x+1
x	0	x	1	x+1
x+1	0	x+1	x+1	0

It is interesting to note that $F[x]/(x^2 + x + 1)$ is actually a field as the multiplicative inverse for all the non-zero elements also exists. On the other hand, $F[x]/(x^2 + 1)$ is *not* a field because the multiplicative inverse of element $x + 1$ does not exist.

It is worthwhile exploring the properties of $f(x)$ which makes $F[x]/f(x)$ a field. As we shall shortly find out, the polynomial $f(x)$ must be *irreducible* (*non-factorizable*).

Definition 4.5 A polynomial $f(x)$ in $F[x]$ is said to be **reducible** if $f(x) = a(x)\ b(x)$, where $a(x)$, $b(x)$ are elements of $F(x)$ and deg $a(x)$ and deg $b(x)$ are both smaller than deg $f(x)$. If $f(x)$ is not reducible, it is called **irreducible**. A monic irreducible polynomial of degree at least one is called a **prime polynomial**.

It is helpful to compare a reducible polynomial with a positive integer that can be factorized into a product of prime numbers. Any monic polynomial in $f(x)$ can be factorized uniquely into a product of irreducible monic polynomials (prime polynomials). One way to verify a prime polynomial is by trial and error, testing for all possible factorizations. This would require a computer search. Prime polynomials of every degree exist over every Galois Field.

Theorem 4.1

(i) A polynomial $f(x)$ has a linear factor $(x - a)$ if and only if $f(x) = 0$ where a is a field element.

(ii) A polynomial $f(x)$ in $F[x]$ of degree 2 or 3 over $GF(q)$ is irreducible if and only if $f(a) \neq 0$ for all a in $GF(q)$.

(iii) Over any field, $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + ... + x + 1)$. The second factor may be further reducible.

Proof

(i) If $f(x) = (x - a)\ g(x)$, then obviously $f(a) = 0$. On the other hand, if $f(a) = 0$, by division algorithm, $f(x) = q(x)(x - a) + r(x)$, where deg $r(x) <$ deg $(x - a) = 1$. This implies that $r(x)$ is a constant. But, since $f(a) = 0$, $r(x)$ must be zero, and therefore, $f(x) = q(x)(x - a)$.

(ii) A polynomial of degree 2 or 3 over $GF(q)$ will be reducible if, and only if, it has at least one linear factor. The result (ii) then directly follows from (i). This result does not necessarily hold for polynomials of degree more than 3. This is because it might be possible to factorize a polynomial of degree 4 or higher into a product of polynomials none of which are linear, i.e., of the type $(x - a)$.

(iii) From (i), $(x - 1)$ is a factor of $(x^n - 1)$. By carrying out long division of $(x^n - 1)$ by $(x - 1)$ we obtain $(x^{n-1} + x^{n-2} + ... + x + 1)$.

Example 4.9 Consider $f(x) = x^3 - 1$ over $GF(2)$. Using (iii) of theorem 4.1 we can write $x^3 - 1 = (x - 1)(x^2 + x + 1)$. This factorization is true over any field. Now, lets try to factorize the second term, $p(x) = (x^2 + x + 1)$.

$$p(0) = 0 + 0 + 1 = 1, \text{ over } GF(2),$$

$$p(1) = 1 + 1 + 1 = 1, \text{ over } GF(2).$$

Therefore, $p(x)$ cannot be factorized further (from Theorem 4.1 (ii)).

Thus, over $GF(2)$, $x^3 - 1 = (x - 1)(x^2 + x + 1)$.

Next, consider $f(x) = x^3 - 1$ over $GF(3)$.

$$x^3 - 1 = (x - 1)(x^2 + x + 1).$$

Again, let $p(x) = (x^2 + x + 1)$.

$$p(0) = 0 + 0 + 1 = 1, \text{ over } GF(3),$$

$$p(1) = 1 + 1 + 1 = 0, \text{ over } GF(3).$$

$$p(2) = 2.2 + 2 + 1 = 1 + 2 + 1 = 1, \text{ over } GF(3).$$

Since, $p(1) = 0$, from (i) we have $(x - 1)$ as a factor of $p(x)$.

Thus, over $GF(3)$,

$$x^3 - 1 = (x - 1)(x - 1)\ (x - 1).$$

Theorem 4.2 The ring $F[x]/f(x)$ is a field if, and only if, $f(x)$ is a prime polynomial in $F[x]$.

Proof To prove that a ring is a field, we must show that every non zero element of the ring has a multiplicative inverse. Let $s(x)$ be a non zero element of the ring. We have, deg $s(x) <$ deg $f(x)$, because $s(x)$ is contained in the ring $F[x]/f(x)$. It can be shown that the Greatest Common Divisor (GCD) of two polynomials $f(x)$ and $s(x)$ can be expressed as

$$\text{GCD}(f(x), s(x)) = a(x)\ f(x) + b(x)\ s(x),$$

where $a(x)$ and $b(x)$ are polynomials over $GF(q)$. Since $f(x)$ is irreducible in $F[x]$, we have

$$\text{GCD}(f(x), s(x)) = 1 = a(x)\, f(x) + b(x)\, s(x).$$

$$\begin{aligned}
\text{Now, } 1 = R_{f(x)}[1] &= R_{f(x)}[\, a(x)\, f(x) + b(x)\, s(x)] \\
&= R_{f(x)}[\, a(x)\, f(x)] + R_{f(x)}[b(x)\, s(x)] && \text{(property (i) of residues)} \\
&= 0 + R_{f(x)}[b(x)\, s(x)] \\
&= R_{f(x)}\{R_{f(x)}[b(x)].R_{f(x)}[s(x)]\} && \text{(property (ii) of residues)} \\
&= R_{f(x)}\{R_{f(x)}[b(x)].s(x)\}
\end{aligned}$$

Hence, $R_{f(x)}[b(x)]$is the multiplicative inverse of $s(x)$.

Next, let us prove the *only if* part of the theorem. Let us suppose $f(x)$ has a degree of at least 2, and is *not* a prime polynomial (a polynomial of degree one is always irreducible). Therefore, we can write

$$f(x) = r(x)\, s(x)$$

for some polynomials $r(x)$ and $s(x)$ with degrees at least one. If the ring $F[x]/f(x)$ is indeed a field, then a multiplicative index of $r(x)$, $r^{-1}(x)$ exists, since all polynomials in the field must have their corresponding multiplicative inverses. Hence,

$$\begin{aligned}
s(x) &= R_{f(x)}\{\, s(x)\} \\
&= R_{f(x)}\{\, r(x) r^{-1}(x) s(x)\} = R_{f(x)}\{r^{-1}(x) r(x) s(x)\} = R_{f(x)}\{r^{-1}(x) f(x)\} = 0
\end{aligned}$$

However, we had assumed $s(x) \neq 0$. Thus, there is a contradiction, implying that the ring is not a field.

Note that a prime polynomial is both monic and irreducible. In the above theorem, it is sufficient to have $f(x)$ irreducible in order to obtain a field. The theorem could as well been stated as: "The ring $F[x]/f(x)$ is a field if and only if $f(x)$ is irreducible in $F[x]$".

So, now we have an elegant mechanism of generating Galois Fields! If we can identify a prime polynomial of degree n over $GF(q)$, we can construct a Galois Field with q^n elements. Such a field will have polynomials as the elements of the field. These polynomials will be defined over $GF(q)$ and consist of all polynomials of degree less than n. It can be seen that there will be q^n such polynomials, which form the elements of the **Extension Field**.

Example 4.10 **Consider the polynomial $p(x) = x^3 + x + 1$ over $GF(2)$. Since, $p(0) \neq 0$ and $p(1) \neq 0$, the polynomial is irreducible in $GF(2)$. Since it is also monic, $p(x)$ is a prime polynomial. Here we have $n = 3$, so we can use $p(x)$ to construct a field with $2^3 = 8$ elements, i.e., $GF(8)$. The elements of this field will be $0, 1, x, x + 1, x^2, x^2 + 1, x^2 + x, x^2 + x + 1$, which are all possible polynomials of degree less than $n = 3$. It is easy to construct the addition and multiplication tables for this field (exercise).**

Having developed the necessary mathematical tools, we now resume our study of cyclic codes. We now fix $f(x) = x^n - 1$ for the remainder of the chapter. We also denote $F[x]/f(x)$ by R_n. Before we proceed, we make the following observations:

(i) $x^n = 1 \pmod{x^n - 1}$. Hence, any polynomial, modulo $x^n - 1$, can be reduced simply by replacing x^n by 1, x^{n+1} by x and so on.

(ii) A codeword can *uniquely* be represented by a polynomial. A codeword consists of a sequence of elements. We can use a polynomial to represent the locations and values of all the elements in the codeword. For example, the codeword $c_1 c_2 ... c_n$ can be represented by the polynomial $c(x) = c_0 + c_1 x + c_2 x^2 + ... c_n x^n$. As another example, the codeword over $GF(8)$, $c = 207735$ can be represented by the polynomial $c(x) = 2 + 7x^2 + 7x^3 + 3x^4 + 5x^5$.

(iii) Multiplying any polynomial by x corresponds to a single cyclic right-shift of the codeword elements. More explicitly, in R_n, by multiplying $c(x)$ by x we get $x.\, c(x) = c_0 x + c_1 x^2 + c_2 x^3 + ... c_n x^{n+1} = c_n + c_0 x + c_1 x^2 + c_2 x^3 + ... c_{n-1} x^n$.

Theorem 4.3 A code C in R_n is a cyclic code if, and only if, C satisfies the following conditions:

(i) $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$ (4.4)

(ii) $a(x) \in C$ and $r(x) \in R_n \Rightarrow a(x) r(x) \in C$. (4.5)

Proof

(i) Suppose C is a cyclic code in R_n. Since cyclic codes are a subset of linear block codes, the first condition holds.

(ii) Let $r(x) = r_0 + r_1 x + r_2 x^2 + ... r_n x^n$. Multiplication by x corresponds to a cyclic rightshift. But, by definition, the cyclic shift of a cyclic codeword is also a valid codeword. That is,

$$x.a(x) \in C,\ x.(xa(x)) \in C,$$

and so on. Hence

$$r(x) a(x) = r_0 a(x) + r_1 x a(x) + r_2 x^2 a(x) + ... r_n x^n a(x)$$

is also in C since each summand is also in C.

Next, we prove the *only if* part of the theorem. Suppose (i) and (ii) hold. Take $r(x)$ to be a scalar. Then (i) implies that C is linear. Take $r(x) = x$ in (ii), which shows that any cyclic shift also leads to a codeword. Hence (i) and (ii) imply that C is a cyclic code.

In the next section, we shall use the mathematical tools developed so far to construct cyclic codes.

4.4 A METHOD FOR GENERATING CYCLIC CODES

The following steps can be used to generate a cyclic code:

(i) Take a polynomial $f(x)$ in R_n.

(ii) Obtain a set of polynomials by multiplying $f(x)$ by all possible polynomials in R_n.

(iii) The set of polynomials obtained above corresponds to the set of codewords belonging to a cyclic code. The blocklength of the code is n.

Example 4.11 Consider the polynomial $f(x) = 1 + x^2$ in R_3 defined over $GF(2)$. In general a polynomial in R_3 ($= F[x]/(x^3 - 1)$) can be represented as $r(x) = r_0 + r_1x + r_2x^2$, where the coefficients can take the values 0 or 1 (since defined over $GF(2)$).

Thus, there can be a total of $2 \times 2 \times 2 = 8$ polynomials in R_3 defined over $GF(2)$, which are 0, 1, x, x^2, $1 + x$, $1 + x^2$, $x + x^2$, $1 + x + x^2$. To generate the cyclic code, we multiply $f(x)$ with these 8 possible elements of R_3 and then reduce the results modulo $(x^3 - 1)$:

$(1 + x^2).0 = 0$, $(1 + x^2).1 = (1 + x^2)$, $(1 + x^2).x = 1 + x$, $(1 + x^2).x^2 = x + x^2$,

$(1 + x^2).(1 + x) = x + x^2$, $(1 + x^2).(1 + x^2) = 1 + x$, $(1 + x^2).(x + x^2) = (1 + x^2)$,

$(1 + x^2).(1 + x + x^2) = 0$.

Thus there are only four distinct codewords: $\{0, 1 + x, 1 + x^2, x + x^2\}$ which correspond to $\{000, 110, 101, 011\}$.

From the above example it appears that we can have some sort of a **Generator Polynomial** which can be used to construct the cyclic code.

Theorem 4.4 Let $\boldsymbol{C}$ be an (n, k) non-zero cyclic code in R_n. Then,

(i) there exists a unique monic polynomial $g(x)$ of the smallest degree in $\boldsymbol{C}$

(ii) the cyclic code $\boldsymbol{C}$ consists of all multiples of the generator polynomial $g(x)$ by polynomials of degree $k - 1$ or less

(iii) $g(x)$ is a factor of $x^n - 1$

Proof

(i) Suppose both $g(x)$ and $h(x)$ are monic polynomials in $\boldsymbol{C}$ of smallest degree. Then $g(x) - h(x)$ is also in $\boldsymbol{C}$, and has a smaller degree. If $g(x) \neq h(x)$, then a suitable scalar multiplier of $g(x) - h(x)$ is monic, and is in $\boldsymbol{C}$, and is of smaller degree than $g(x)$. This gives a contradiction.

(ii) Let $a(x) \in \boldsymbol{C}$. Then, by division algorithm, $a(x) = q(x)g(x) + r(x)$, where $\deg r(x) < \deg g(x)$. But $r(x) = a(x) - q(x)g(x) \in \boldsymbol{C}$ because both words on the right hand side of the equation are codewords. However, the degree of $g(x)$ must be the minimum among all codewords. This can only be possible if $r(x) = 0$ and $a(x) = q(x)g(x)$. Thus, a codeword is obtained by multiplying the generator polynomial $g(x)$ with the polynomial $q(x)$. For a code defined over $GF(q)$, here are q^k distinct codewords possible. These codewords correspond to multiplying $g(x)$ with the q^k distinct polynomials, $q(x)$, where $\deg q(x) \leq (k - 1)$.

(iii) By division algorithm, $x^n - 1 = q(x)g(x) + r(x)$, where $\deg r(x) < \deg g(x)$. Or, $r(x) = \{(x^n - 1) - q(x)g(x)\}$ modulo $(x^n - 1) = - q(x)g(x)$. But $- q(x)g(x) \in \boldsymbol{C}$ because we are multiplying the generator polynomial by another polynomial $-q(x)$. Thus, we have a codeword $r(x)$ whose degree is less than that of $g(x)$. This violates the minimality of the degree of $g(x)$, unless $r(x) = 0$. Which implies $x^n - 1 = q(x)\, g(x)$, i.e., $g(x)$ is a factor of $x^n - 1$.

The last part of the theorem gives us the recipe to obtain the generator polynomial for a cyclic code. All we have to do is to factorize $x^n - 1$ into irreducible, monic polynomials. We can also find all the possible cyclic codewords of blocklength n simply by factorizing $x^n - 1$.

Note 1: A cyclic code $\boldsymbol{C}$ may contain polynomials other than the generator polynomial which *also* generate $\boldsymbol{C}$. But the polynomial with the minimum degree is called the generator polynomial.

Note 2: The degree of $g(x)$ is $n - k$ (this will be shown later).

Example 4.12 To find all the binary cyclic codes of blocklength 3, we first factorize $x^3 - 1$. Note that for $GF(2)$, $1 = -1$, since $1 + 1 = 0$. Hence,

$$x^3 - 1 = x^3 + 1 = (x + 1)(x^2 + x + 1)$$

Thus, we can make the following table.

Generator Polynomial	Code (polynomial)	Code (binary)
1	$\{R_3\}$	{000, 001, 010, 011, 100, 101, 110, 111}
$(x + 1)$	$\{0, x + 1, x^2 + x, x^2 + 1\}$	{000, 011, 110, 101}
$(x^2 + x + 1)$	$\{0, x^2 + x + 1\}$	{000, 111}
$(x^3 + 1) = 0$	$\{0\}$	{000}

A simple encoding rule to generate the codewords from the generator polynomial is

$$c(x) = i(x)\, g(x), \tag{4.6}$$

where $i(x)$ is the information polynomial, $c(x)$ is the codeword polynomial and $g(x)$ is the generator polynomial. We have seen, already, that there is a one to one correspondence between a word (vector) and a polynomial. The error vector can be also represented as the error polynomial, $e(x)$. Thus, the received word at the receiver, after passing through a noisy channel can be expressed as

$$v(x) = c(x) + e(x). \tag{4.7}$$

We define the **Syndrome Polynomial**, $s(x)$ as the remainder of $v(x)$ under division by $g(x)$, i.e.,

$$s(x) = R_{g(x)}[v(x)] = R_{g(x)}[\, c(x) + e(x)] = R_{g(x)}[c(x)] + R_{g(x)}[e(x)] = R_{g(x)}[e(x)], \tag{4.8}$$

because $R_{g(x)}[c(x)] = 0$.

Example 4.13 Consider the generator polynomial $g(x) = x^2 + 1$ for ternary cyclic codes (*i.e.*, over $GF(3)$) of blocklength $n = 4$. Here we are dealing with cyclic codes, therefore, the highest power of $g(x)$ is $n - k$. Since $n = 4$, k must be 2. So, we are going to construct a (4, 2) cyclic ternary code. There will be a total of $q^k = 3^2 = 9$ codewords. The information polynomials and the corresponding codeword polynomials are listed below.

i	$i(x)$	$c(x) = i(x)\,g(x)$	c
00	0	0	0000
01	1	$x^2 + 1$	0101
02	2	$2x^2 + 2$	0202
10	x	$x^3 + x$	1010
11	$x + 1$	$x^3 + x^2 + x + 1$	1111
12	$x + 2$	$x^3 + 2x^2 + x + 2$	1212
20	$2x$	$2x^3 + 2x$	2020
21	$2x + 1$	$2x^3 + x^2 + 2x + 1$	2121
22	$2x + 2$	$2x^3 + 2x^2 + 2x + 2$	2222

It can be seen that the cyclic shift of any codeword results in another valid codeword. By observing the codewords we find that the minimum distance of this code is 2 (there are four non-zero codewords with the minimum Hamming weight = 2). Therefore, this code is capable of detecting one error and correcting zero errors.

Observing the fact that the codeword polynomial is divisible by the generator polynomial, we can *detect* more number of errors than suggested by the minimum distance of the code. Since we are dealing with cyclic codes that are a subset of linear block codes, we can use the all zero codeword to illustrate this point without loss of generality.

Assume that $g(x) = x^2 + 1$ and the transmitted codeword is the all zero codeword.

Therefore, the received word is the error polynomial, i.e.,

$$v(x) = c(x) + e(x) = e(x). \tag{4.9}$$

At the receiver end, an error will be detected if $g(x)$ fails to divide the received word $v(x) = e(x)$. Now, $g(x)$ has only two terms. So if the $e(x)$ has odd number of terms, i.e., if the number of errors are odd, it will be caught by the decoder! For example, if we try to divide $e(x) = x^3 + x + 1$ by $g(x)$, we will always get a remainder. In the example of the (4, 2) cyclic code with $g(x) = x^2 + 1$, the $d^* = 2$, suggesting that it can detect $d^* - 1 = 1$ error. However, by this simple observation, we find that it can detect any odd number of errors $\leq n$. In this case, it can detect 1 error or 3 errors, but not 2 errors.

4.5 MATRIX DESCRIPTION OF CYCLIC CODES

Theorem 4.5 Suppose C is a cyclic code with generator polynomial $g(x) = g_0 + g_1x + \ldots + g_rx^r$ of degree r. Then the generator matrix of C is given by

$$G = \underbrace{\begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & \cdots & g_r \end{bmatrix}}_{n \text{ columns}} \Bigg\downarrow k = (n - r) \text{ rows} \tag{4.10}$$

Proof The $(n - r)$ rows of the matrix are obviously linearly independent because of the echelon form of the matrix. These $(n - r)$ rows represent the codewords $g(x)$, $xg(x)$, $x^2 g(x)$, ..., $x^{n-r-1}g(x)$. Thus, the matrix can generate these codewords. Now, to prove that the matrix can generate all the possible codewords, we must show that every possible codeword can be represented as linear combinations of the codewords $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$.

We know that if $c(x)$ is a codeword, it can be represented as

$$c(x) = q(x).g(x)$$

for some polynomial $q(x)$. Since the degree of $c(x) < n$ (because the length of the codeword is n), it follows that the degree of $q(x) < n - r$. Hence,

$$q(x).g(x) = (q_0 + q_1x + \ldots + q_{n-r-1}x^{n-r-1})g(x) = q_0g(x) + q_1xg(x) + \ldots + q_{n-r-1}x^{n-r-1}g(x)$$

Thus, any codeword can be represented as a linear combination of $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$. This proves that the matrix G is indeed the generator matrix.

We also know that the dimensions of the generator matrix is $k \times n$. Therefore, $r = n - k$, i.e., the degree of $g(x)$ is $n - k$.

Example 4.14 To find the generator matrices of all ternary codes (i.e., codes over $GF(3)$) of blocklength $n = 4$, we first factorize $x^4 - 1$.

$$x^4 - 1 = (x - 1)(x^3 + x^2 + x + 1) = (x - 1)(x + 1)(x^2 + 1)$$

We know that all the factors of $x^4 - 1$ are capable of generating a cyclic code. The resultant generator matrices are listed in Table 4.1. Note that $-1 = 2$ for $GF(3)$.

Table 4.1 *Cyclic codes of blocklength n = 4 over GF(3)*

$g(x)$	(n, k)	d_{min}	G
1	(4, 4)	1	$[I_4]$
$(x - 1)$	(4, 3)	2	$\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$

$g(x)$	(n, k)	d_{min}	G
$(x+1)$	(4, 3)	2	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$
(x^2+1)	(4, 2)	2	$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
(x^2-1)	(4, 2)	2	$\begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$
(x^3-x^2+x-1)	(4,1)	4	[−1 1 −1 1]
(x^3+x^2+x+1)	(4, 1)	4	[+1 1 +1 1]
(x^4-1)	(4, 1)	0	[0 0 0 0]

It can be seen from the table that none of the (4, 2) ternary cyclic codes are single error correcting codes (since their minimum distance is less than 3). An interesting observation is that we do not have any ternary (4, 2) Hamming Code that is cyclic! Remember, Hamming Codes are single error correcting codes with $n = (q^r - 1)/(q - 1)$ and $k = (q^r - 1)/(q - 1) - r$, where r is an integer ≥ 2. Therefore, a (4, 2) ternary Hamming code exists, but it is not a cyclic code.

The next step is to explore if we can find a parity check polynomial corresponding to our generator polynomial, $g(x)$. We already know that $g(x)$ is a factor of $x^n - 1$. Hence we can write

$$x^n - 1 = h(x)\, g(x), \tag{4.11}$$

where $h(x)$ is some polynomial. The following can be concluded by simply observing the above equation:

(i) Since $g(x)$ is monic, $h(x)$ has to be monic because the left hand side of the equation is also monic (the leading coefficient is unity).

(ii) Since degree of $g(x)$ is $n - k$, the degree of $g(x)$ must be k.

Suppose $\boldsymbol{C}$ is a cyclic code in R_n with the generator polynomial $g(x)$. Recall that we are den-oting $F[x]/f(x)$ by R_n, where $f(x) = x^n - 1$. In R_n, $h(x)g(x) = x^n - 1 = 0$. Then, any codeword belo-nging to $\boldsymbol{C}$ can be written as $c(x) = a(x)g(x)$, where the polynomial $a(x) \in R_n$. Therefore in R_n,

$$c(x)h(x) = a(x)g(x)h(x) = \boldsymbol{a}(\boldsymbol{x}) \cdot 0 = 0.$$

Thus, $h(x)$ behaves like a **Parity Check Polynomial**. Any valid codeword when multiplied by the parity check polynomial yields the zero polynomial. This concept is parallel to that of the parity check matrix introduced in the previous chapter. Since we are still in the domain of linear block codes, we go ahead and define the parity check matrix in relation to the parity check polynomial.

Suppose $\boldsymbol{C}$ is a cyclic code with the parity check polynomial $h(x) = h_0 + h_1 x + \ldots + h_k x^k$, then the parity check matrix of $\boldsymbol{C}$ is given by

$$\boldsymbol{H} = \underbrace{\begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix}}_{n \text{ columns}} \quad (n = k) \text{ rows} \tag{4.12}$$

Recall that $\boldsymbol{c}\boldsymbol{H}^T = \boldsymbol{0}$. Therefore, $\boldsymbol{i}\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$ for any information vector, $\boldsymbol{i}$. Hence, $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$. We further have $\boldsymbol{s} = \boldsymbol{v}\boldsymbol{H}^T$ where $\boldsymbol{s}$ is the syndrome vector and $\boldsymbol{v}$ is the received word.

Example 4.15 For binary codes of block length, $n = 7$, we have

$$x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1)$$

Consider $g(x) = (x^3 + x + 1)$. Since $g(x)$ is a factor of $x^7 - 1$, there is a cyclic code that can be generated by it. The generator matrix corresponding to $g(x)$ is

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The parity check polynomial $h(x)$ is $(x - 1)(x^3 + x^2 + 1) = (x^4 + x^2 + x + 1)$. And the corresponding parity check matrix is

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The minimum distance of this code is 3, and this happens to be the (7, 4) Hamming Code. Thus, the binary (7, 4) Hamming Code is also a Cyclic code.

4.6 BURST ERROR CORRECTION

In many real life channels, errors are not random, but occur in bursts. For example, in a mobile communications channel, fading results in Burst errors. When errors occur at a stretch, as opposed to random errors, we term them as **Burst Errors**.

Example 4.16 Let the transmitted sequence of bits, transmitted at 10 kb/s over a wireless channel, be

$$c = 0\,1\,0\,0\,0\,1\,1\,1\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1$$

Suppose, after 0.5 ms of the start of transmission, the channel experiences a fade of duration 1 ms. During this time interval, the channel corrupts the transmitted bits. The error sequence can be written as

$$b = 0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0.$$

This is an example of a burst error, where a portion of the transmitted sequence gets garbled due to the channel. Here the *length* of the burst is 10 bits. However, not all ten locations are in error.

Definition 4.6 A Cyclic Burst of length t is a vector whose non-zero components are among t successive components, the first and last of which are non-zero.

If we are constructing codes for channels more prone to burst errors of length t (as opposed to an arbitrary pattern of t random errors), it might be possible to design more efficient codes. We can describe a burst error as

$$e(x) = x^i . b(x) \tag{4.13}$$

where is a polynomial of degree $\leq t-1$ and $b(x)$ is the burst pattern. x^i marks the starting location of the burst pattern within the codeword being transmitted.

A code designed for correcting a burst of length t must have unique syndromes for every error pattern, i.e.,

$$s(x) = R_{g(x)}[e(x)] \tag{4.14}$$

is different for each polynomial representing a burst of length t.

Example 4.17 For a binary code of blocklength $n = 15$, consider the generator polynomial

$$g(x) = x^6 + x^3 + x^2 + x + 1$$

This code is capable of correcting bursts of length 3 or less. To prove this we must show that all the syndromes corresponding to the different burst errors are distinct. The different Burst Errors are

(i) Bursts of length 1

$$e(x) = x^i \text{ for } i = 0, 1, ..., 14.$$

(ii) Bursts of length 2

$$e(x) = x^i \cdot (1 + x) \quad \text{for } i = 0, 1, ..., 13, \text{ and } e(x) = x^i \cdot (1 + x^2) \quad \text{for } i = 0, 1, ..., 13.$$

(iii) Bursts of length 3

$$e(x) = x^i \cdot (1 + x + x^2) \quad \text{for } i = 0, 1, ..., 12.$$

It can be shown that the syndrome of all these 56 (15 + 14 + 14 + 13) error patterns are distinct. A table can be made for each pattern and the corresponding syndrome which can be used for correcting a burst error of length 3 or less. It should be emphasized that the codes designed specifically for correcting burst errors are more efficient in terms of the code rate. The code being discussed here is a (15, 9) cyclic code with code rate $= k/n = 0.6$ and minimum distance $d^* = 3$. This code can correct only 1 random error (but up to three burst errors!). Note that correction of one random error amounts to correcting a burst error of length 1.

Similar to the Singleton Bound studied in the previous chapter, there is a Bound for the minimum number of parity bits required for a burst-error correcting linear block code: 'A linear block code that corrects all bursts of length t or less must have at least $2t$ parity symbols'.

In the next three sections, we will study three different sub-classes of cyclic codes. Each sub-class has a specific objective.

4.7 FIRE CODES

Definition 4.7 A Fire code is a cyclic burst error correcting code over $GF(q)$ with the generator polynomial

$$g(x) = (x^{2t-1}-1)p(x), \tag{4.15}$$

where $p(x)$ is a prime polynomial over $GF(q)$ whose degree m is not smaller than t and $p(x)$ does not divide $x^{2t-1}-1$. The blocklength of the Fire Code is the smallest integer n such that $g(x)$ divides x^n-1. A Fire Code can correct all burst errors of length t or less.

Example 4.18 Consider the Fire code with $t = m = 3$. A prime polynomial over $GF(2)$ of degree 3 is $p(x) = x^3 + x + 1$, which does not divide (x^5-1). The generator polynomial of the Fire Code will be

$$\begin{aligned} g(x) &= (x^5-1)p(x) = (x^5-1)(x^3 + x + 1) \\ &= x^8 + x^6 + x^5 - x^3 - x - 1 \\ &= x^8 + x^6 + x^5 + x^3 + x + 1 \end{aligned}$$

The degree of $g(x) = n - k = 8$. The blocklength is the smallest integer n such that $g(x)$ divides x^n-1. After trial and error we get $n = 35$. Thus, the parameters of the Fire Code are (35, 27) with $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$. This code can correct up to bursts of length 3. The code rate of this code is 0.77, and is more efficient than the code generated by $g(x) = x^6 + x^3 + x^2 + x + 1$ which has a code rate of only 0.6.

Fire Codes become more efficient as we increase t. The code rates for binary Fire Codes (with $m = t$) for different values of t are plotted in Fig. 4.1.

Fig 4.1 *Code Rates for Different Fire Codes.*

4.8 GOLAY CODES

The Binary Golay Code

In the previous chapter, Sec. 3.9, we saw that a (23, 12) perfect code exists with $d^* = 7$. Recall that, for a perfect code,

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\cdots+\binom{n}{t}(q-1)^t\right\}=q^n, \tag{4.16}$$

which is satisfied for the values: $n = 23$, $k = 12$, $M = 2^k = 2^{12}$, $q = 2$ and $t = (d^* - 1)/2 = 3$. This (23, 12) perfect code is the Binary Golay Code. We shall now explore this Perfect Code as a cyclic code. We start with the factorization of $(x^{23}-1)$.

$$\begin{aligned}(x^{23}-1) &= (x-1)(x^{11}+x^{10}+x^6+x^5+x^4+x^2+1)\,(x^{11}+x^9+x^7+x^6+x^5+x+1)\\ &= (x-1)\,g_1(x)\,g_2(x).\end{aligned} \tag{4.17}$$

The degree of $g_1(x) = n - k = 11$, hence $k = 12$, which implies that there exists a (23, 12) cyclic code. In order to prove that it is a perfect code, we must show that the minimum distance of this (23, 12) cyclic code is 7. One way is to write out the parity check matrix, $\boldsymbol{H}$, and show that no six columns are linearly dependent. Another way is to prove it analytically, which is a long and drawn-out proof. The easiest way is to write a computer program to list out all the 2^{12} codewords and find the minimum weight (on a fast computer it takes about 30 seconds!). The code rate is 0.52 and it is a triple error correcting code. However, the relatively small block length of this perfect code makes it impractical for most real life applications.

The Ternary Golay Code

We next examine the ternary (11, 6) cyclic code, which is also the Ternary Golay Code. This code has a minimum distance = 5, and can be verified to be a perfect code. We begin by factorizing $(x^{11}-1)$ over $GF(3)$.

$$\begin{aligned}(x^{11}-1) &= (x-1)(x^5+x^4-x^3+x^2-1)\,(x^5-x^3-x^2-x-1)\\ &= (x-1)\,g_1(x)\,g_2(x)\end{aligned} \tag{4.18}$$

The degree of $g_1(x) = n - k = 5$, hence $k = 6$, which implies that there exists a (11, 6) cyclic code. In order to prove that it is a perfect code, we must show that the minimum distance of this (11, 6) cyclic code is 5. Again, we resort to an exhaustive computer search and find that the minimum distance is indeed 5.

It can be shown that (x^p-1) has a factorization of the form $(x - 1)\,g_1(x)\,g_2(x)$ over $GF(2)$, whenever p is a prime number of the form $8m \pm 1$ (m is a positive integer). In such cases, $g_1(x)$ and $g_2(x)$ generate equivalent codes. If the minimum distance of the code generated by $g_1(x)$ is *odd*, then it satisfies the **Square Root Bound**

$$d^* \geq \sqrt{p} \tag{4.19}$$

Note that p denotes the blocklength.

4.9 CYCLIC REDUNDANCY CHECK (CRC) CODES

One of the common error detecting codes is the Cyclic Redundancy Check (CRC) Codes. For a k-bit block of bits the (n, k) CRC encoder generates $(n - k)$ bit long frame check sequence (FCS). Let us define the following

T = n-bit frame to be transmitted

D = k-bit message block (information bits)

F = $(n - k)$ bit FCS, the last $(n - k)$ bits of T

P = the predetermined divisor, a pattern of $(n - k + 1)$ bits.

The pre-determined divisor, P, should be able to divide the codeword T. Thus, T/P has no remainder. Now, D is the k-bit message block. Therefore, $2^{n-k}D$ amounts to shifting the k bits to the left by $(n - k)$ bits, and padding the result with zeros (recall that left shift by 1 bit of a binary sequence is equivalent to multiplying the number represented by the binary sequence by two). The codeword, T, can then be represented as

$$T = 2^{n-k}D + F \tag{4.20}$$

Adding F in the above equation yields the concatenation of D and F. If we divide $2^{n-k}D$ by P, we obtain

$$\frac{2^{n-k}D}{P} = Q + \frac{R}{P} \tag{4.21}$$

where, Q is the quotient and R/P is the remainder. Suppose we use R as the FCS, then,

$$T = 2^{n-k}D + R \tag{4.22}$$

In this case, upon dividing T by P we obtain

$$\frac{T}{P} = \frac{2^{n-k}D+R}{P} = \frac{2^{n-k}D}{P} + \frac{R}{P}$$

$$= Q + \frac{R}{P} + \frac{R}{P} = Q + \frac{R+R}{P} = Q \tag{4.23}$$

Thus there is no remainder, i.e., T is exactly divisible by P. To generate such an FCS, we simply divide $2^{n-k}D$ by P and use the $(n-k)$-bit remainder as the FCS.

Let an error E occur when T is transmitted over a noisy channel. The received word is given by

$$V = T + E \tag{4.24}$$

The CRC scheme will fail to detect the error only if V is completely divisible by P. This translates to the case when E is completely divisible by P (because T is divisible by P).

Example 4.19 Let the message $D = 1010001101$, i.e., $k = 10$ and the pattern, $P = 110101$. The number of FCS bits = 5. Therefore, $n = 15$. We wish to determine the FCS.

First, the message is multiplied by 2^5 (left shift by 5 and pad with 5 zeros). This yields

$$2^5 D = 101000110100000$$

Next divide the resulting number by $P = 110101$. By long division, we obtain $Q = 1101010110$ and $R = 01110$. The remainder is added to 2^5D to obtain

$$T = 101000110101110$$

T is the transmitted codeword. If no errors occur in the channel, the received word when divided by P will yield 0 as the remainder.

CRC codes can also be defined using the polynomial representation. Let the message polynomial be $D(x)$ and the predetermined divisor be $P(x)$. Therefore,

$$\frac{x^{n-k}D(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)}$$

$$T(x) = x^{n-k}D(x) + R(x) \tag{4.25}$$

At the receiver end, the received word is divided by $P(x)$. Suppose the received word is

$$V(x) = T(x) + E(x), \tag{4.26}$$

where $E(x)$ is the error polynomial. Then $[T(x) + E(x)]/P(x) = E(x)/P(x)$ because $T(x)/P(x) = 0$. Those errors that happen to correspond to polynomials containing $P(x)$ as a factor will slip by, and the others will be caught in the net of the CRC decoder. The polynomial $P(x)$ is also called the generator polynomial for the CRC code. CRC codes are also known as **Polynomial Codes**.

Example 4.20 Suppose the transmitted codeword undergoes a single-bit error. The error polynomial $E(x)$ can be represented by $E(x) = x^i$, where i determines the location of the single error bit. If $P(x)$ contains two or more terms, $E(x)/P(x)$ can never be zero. Thus all the single errors will be caught by such a CRC code.

Example 4.21 Suppose two isolated errors occur, i.e., $E(x) = x^i + x^j$, $i > j$. Alternately, $E(x) = x^j(x^{i-j} + 1)$. If we assume that $P(x)$ is not divisible by x, then a sufficient condition for detecting all double errors is that $P(x)$ does not divide $x^k + 1$ for any k up to the maximum value of $i - j$ (i.e., the frame length). For example $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any value of k below 32,768.

Example 4.22 Suppose the error polynomial has an odd number of terms (corresponding to an odd number of errors). An interesting fact is that there is no polynomial with an odd number of terms that has $x + 1$ as a factor if we are performing binary arithmetic (modulo 2 operations). By making $(x + 1)$ as a factor of $P(x)$, we can catch all errors consisting of odd number of bits (i.e., we can catch at least half of all possible errors!).

Another interesting feature of CRC codes is its ability to detect burst errors. A burst error of length k can be represented by $x^i(x^{k-1} + x^{k-2} + \ldots + 1)$, where i determines how far from the right end of the received frame the burst is located. If $P(x)$ has a x^0 term, it will not have an x^i as a factor. So, if the degree of $(x^{k-1} + x^{k-2} + \ldots + 1)$ is less than the degree of $P(x)$, the remainder can never be zero. Therefore, a polynomial code with r check bits can detect all burst errors of length $\leq r$. If the burst length is $r + 1$, the remainder of the division by $P(x)$ will be zero if, and only if, the burst is identical to $P(x)$. Now, the 1st and last bits of a burst must be 1 (by definition). The intermediate bits can be 1 or 0. Therefore, the exact matching of the burst error with the polynomial $P(x)$ depends on the $r - 1$ intermediate bits. Assuming all combinations are equally likely, the probability of a miss is $\frac{1}{2^{r-1}}$. One can show that when error bursts of length greater than $r + 1$ occurs, or several shorter bursts occur, the probability of a bad frame slipping through is $\frac{1}{2^r}$.

Example 4.23 Four versions of $P(x)$ have become international standards:

CRC-12: $P(x) = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$.

CRC-16: $P(x) = x^{16} + x^{15} + x^2 + 1$.

CRC-CCITT: $P(x) = x^{16} + x^{15} + x^5 + 1$.

CRC-32: $P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$. (4.27)

All the four contain $(x + 1)$ as a factor. CRC-12 is used for transmission of streams of 6-bit characters and generates a 12-bit FCS. Both CRC-16 and CRC-CCITT are popular for 8-bit characters. They result in a 16 bit FCS and can catch all single and double errors, all errors with odd number of bits, all burst errors of length 16 or less, 99.997% of 17-bit bursts and 99.998% of18-bit and longer bursts. CRC-32 is specified as an option in some point-to-point **Synchronous Transmission Standards**.

4.10 CIRCUIT IMPLEMENTATION OF CYCLIC CODES

Shift registers can be used to encode and decode cyclic codes easily. Encoding and decoding of cyclic codes require multiplication and division by polynomials. The shift property of shift registers are ideally suited for such operations. Shift registers are banks of memory units which are capable of shifting the contents of one unit to the next at every clock pulse. Here we will focus on circuit implementation for codes over $GF(2^m)$. Beside the shift register, we will make use of the following circuit elements:

(i) A scaler, whose job is to multiply the input by a fixed field element.

(ii) An adder, which takes in two inputs and adds them together. A simple circuit realization of an adder is the *'exclusive-or'* or the *'xor'* gate.

(iii) A multiplier, which is basically the *'and'* gate.

These elements are depicted in Fig. 4.2.

Fig. 4.2 *Circuit Elements Used to Construct Encoders and Decoders for Cyclic Codes.*

A field element of $GF(2)$ can simply be represented by a single bit. For $GF(2^m)$ we require m bits to represent one element. For example, elements of $GF(8)$ can be represented as the elements of the set {000, 001, 010, 011, 100, 101, 110, 111}. For such a representation we need three clock pulses to shift the element from one *stage* of the shift register to the next. The effective shift register for $GF(8)$ is shown in Fig. 4.3. Any arbitrary element of this field can be represented by $ax^2 + bx + c$, where a, b, c are binary, and the power of the indeterminate x is used to denote the position. For example, $101 = x^2 + 1$.

Fig. 4.3 *The Effective Shift Register for GF(8).*

Example 4.24 We now consider the multiplication of any arbitrary element by another field element over $GF(8)$. Recall the construction of $GF(8)$ from $GF(2)$ using the prime polynomial $p(x) = x^3 + x + 1$. The elements of the field will be $0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1$. We want to obtain the circuit representation for the multiplication of any arbitrary field element $(ax^2 + bx + c)$ by another element, say, $x^2 + x$. We have

$$(ax^2 + bx + c)(x^2 + x) = ax^4 + (a+b)x^3 + (b+c)x^2 + cx \qquad (\text{modulo } p(x))$$
$$= (a+b+c)x^2 + (b+c)x + (a+b)$$

One possible circuit realization is shown in Fig. 4.4.

Fig. 4.4 *Multiplication of an Arbitrary Field Element.*

We next focus on the multiplication of an arbitrary polynomial $a(x)$ by $g(x)$. Let the polynomial $g(x)$ be represented as

$$g(x) = g_L x^L + \ldots + g_1 x + g_0, \tag{4.28}$$

the polynomial $a(x)$ be represented as

$$a(x) = a_k x^k + \ldots + a_1 x + a_0, \tag{4.29}$$

the resultant polynomial $b(x) = a(x)g(x)$ be represented as

$$b(x) = b_{k+L} x^{k+L} + \ldots + b_1 x + b_0. \tag{4.30}$$

The circuit realization of $b(x)$ is given in Fig. 4.5. This is linear feed-forward shift register. It is also called a **Finite Impulse Response (FIR) Filter**.

Fig. 4.5 *A Finite Impuse Response (FIR) Filter.*

In Electrical Engineering jargon, the coefficients of $a(x)$ and $g(x)$ are convolved by the shift register. For our purpose, we have a circuit realization for multiplying two polynomials. Thus, we have an efficient mechanism of encoding a cyclic code by multiplying the information polynomial by the generator polynomial.

Example 4.25 The encoder circuit for the generator polynomial

$$g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$$

is given in Fig. 4.6.

This is the generator polynomial for the Fire Code with $t = m = 3$. It is easy to interpret the circuit. The 8 memory units shift the input, one unit at a time. The shifted outputs are summed at the proper locations. There are five adders for summing up the six shifted versions of the input.

Fig. 4.6 *Circuit Realization of the Encoder for the Fire Code.*

For dividing an arbitrary polynomial by a fixed polynomial $g(x)$, the circuit realization is given in Fig. 4.7.

We can also use a shift register circuit for dividing an arbitrary polynomial, $a(x)$, by a fixed polynomial $g(x)$. We assume here that the divisor is a monic polynomial. We already know how to factor out a scalar in order to convert any polynomial to a monic polynomial. The division process can be expressed as a pair of recursive equations. Let $Q^{(r)}(x)$ and $R^{(r)}(x)$ be the quotient polynomial and the remainder polynomial at the rth recursion step, with the initial conditions $Q^{(0)}(x) = 0$ and $R^{(0)}(x) = a(x)$. Then, the recursive equations can be written as

$$Q^{(r)}(x) = Q^{(r-1)}(x) + R_{(n-r)}x^{k-r},$$

$$R^{(r)}(x) = R^{(r-1)}(x) - R_{(n-r)}x^{k-r}g(x), \tag{4.31}$$

where $R_{(n-r)}$ represents the leading coefficient of the remainder polynomial at stage $(r-1)$.

For dividing an arbitrary polynomial $a(x)$ by a fixed polynomial $g(x)$, the circuit realization is given in Fig. 4.8. After n shifts, the quotient is passed out of the shift register, and the value stored in the shift register is the remainder. Thus the shift register implementation of a decoder is very simple. The contents of the shift register can be checked for all entries to be zero after the division of the received polynomial by the generator polynomial. If even a single memory unit of the shift register is non-zero, an error is detected.

Fig. 4.7 *A shift Register Circuit for Dividing by g(x).*

Example 4.26 The shift register circuit for dividing by $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$ is given in Fig. 4.8.

Fig. 4.8 *A Shift Register Circuit for Dividing by $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$.*

The procedure for error detection and error correction is as follows. The received word is first stored in a buffer. It is subjected to divide-by-$g(x)$ operation. As we have seen, the division can be carried out very efficiently by a shift register circuit. The remainder in the shift register is then compared with all the possible (pre-computed) syndromes. This set of syndromes corresponds to the set of correctable error patterns. If a syndrome match is found, the error is subtracted out from the received word. The corrected version of the received word is then passed on to the next stage of the receiver unit for further processing. This kind of a decoder is known as **Meggitt Decoder**. The flow chart for this is given in Fig. 4.9.

Fig. 4.9 *The Flow Chart of a Meggitt Decoder.*

4.11 CONCLUDING REMARKS

The notion of cyclic codes was first introduced by Prange in 1957. The work on cyclic codes was further developed by Peterson and Kasami. Pioneering work on the minimum distance of cyclic codes was done by Bose and Raychaudhuri in the early 1960s. Another subclass of cyclic codes, the BCH codes (named after Bose, Chaudhuri and Hocquenghem) will be studied in detail in the next chapter. It was soon discovered that almost all of the earlier discovered linear block codes could be made cyclic. The initial steps in the area of burst error correction was taken by Abramson in 1959. The Fire Codes were published in the same year. The binary and the ternary Golay Codes were published by Golay in, as early as, 1949.

Shift register circuits for cyclic codes were introduced in the works of Peterson, Chien and Meggitt in the early 1960s. Important contributions were also made by Kasami, MacWilliams, Mitchell and Rudolph.

SUMMARY

- A polynomial is a mathematical expression $f(x) = f_0 + f_1x + \ldots + f_mx^m$, where the symbol x is called the indeterminate and the coefficients $f_0, f_1, \ldots f_m$ are the elements of $GF(q)$. The coefficient f_m is called the leading coefficient. If $f_m \neq 0$, then m is called the degree of the polynomial, and is denoted by deg $f(x)$. A polynomial is called monic if its leading coefficient is unity.
- The division algorithm states that, for every pair of polynomial $a(x)$ and $b(x) \neq 0$ in $F[x]$, there exists a unique pair of polynomials $q(x)$, the quotient, and $r(x)$, the remainder, such that $a(x) = q(x)\ b(x) + r(x)$, where deg $r(x) <$ deg $b(x)$. The remainder is sometimes also called the residue, and is denoted by $R_{b(x)}[a(x)] = r(x)$.
- Two important properties of residues are
  (i) $R_{f(x)}[a(x) + b(x)] = R_{f(x)}[a(x)] + R_{f(x)}[b(x)]$, and
  (ii) $R_{f(x)}[a(x).\ b(x)] = R_{f(x)}\{R_{f(x)}[a(x)].\ R_{f(x)}[b(x)]\}$
  where $a(x)$, $b(x)$ and $f(x)$ are polynomials over $GF(q)$.
- A polynomial $f(x)$ in $F[x]$ is said to be reducible if $f(x) = a(x)\ b(x)$, where $a(x)$, $b(x)$ are elements of $f(x)$ and deg $a(x)$ and deg $b(x)$ are both smaller than deg $f(x)$. If $f(x)$ is not reducible, it is called irreducible. A monic irreducible polynomial of degree at least one is called a prime polynomial.
- The ring $F[x]/f(x)$ is a field if and only if $f(x)$ is a prime polynomial in $F[x]$.
- A code C in R_n is a cyclic code if and only if C satisfies the following conditions:
  (i) $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$,
  (ii) $a(x) \in C$ and $r(x) \in R_n \Rightarrow a(x)r(x) \in C$.
- The following steps can be used to generate a cyclic code:
  (i) Take a polynomial $f(x)$ in R_n.

  (ii) Obtain a set of polynomials by multiplying $f(x)$ by all possible polynomials in R_n.
  (iii) The set of polynomials obtained as above corresponds to the set of codewords belonging to a cyclic code. The blocklength of the code is n.
- Let C be a (n, k) non zero cyclic code in R_n. Then,
  (i) there exists a unique monic polynomial $g(x)$ of the smallest degree in C,
  (ii) the cyclic code C consists of all multiples of the generator polynomial $g(x)$ by polynomials of degree $k - 1$ or less.
  (iii) $g(x)$ is a factor of $x^n - 1$.
  (iv) The degree of $g(x)$ is $n - k$.
- For a cyclic code, $\boldsymbol{C}$, with generator polynomial $g(x) = g_0 + g_1x + \ldots + g_rx^r$ of degree r, the generator matrix is given by

$$\boldsymbol{G} = \underbrace{\begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & \cdots & g_r \end{bmatrix}}_{n \text{ columns}} \Big\} k = n - r \text{ rows}$$

- For a cyclic code, $\boldsymbol{C}$, with the parity check polynomial $h(x) = h_0 + h_1x + \ldots + h_kx^k$, then the parity check matrix is given by

$$\boldsymbol{H} = \underbrace{\begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix}}_{n \text{ columns}} \Big\} (n - k) \text{ rows}$$

- $x^n - 1 = h(x)\ g(x)$, where $g(x)$ is the generator polynomial and $h(x)$ is the parity check polynomial.
- A fire code is a cyclic burst error correcting code over $GF(q)$ with the generator polynomial $g(x) = (x^{2t-1}-1)\ p(x)$, where $p(x)$ is a prime polynomial over $GF(q)$ whose degree m is not smaller than t and $p(x)$ does not divide $x^{2t-1}-1$. The blocklength of the Fire Code is the smallest integer n such that $g(x)$ divides x^n-1. A Fire Code can correct all burst errors of length t or less.
- The generator polynomial of the Binary Golay Code:
  $g_1(x) = (x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1)$, or
  $g_2(x) = (x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1)$.

- The generator polynomial of the Ternary Golay Code:
  $g_1(x) = (x^5 + x^4 - x^3 + x^2 - 1)$, or
  $g_2(x) = (x^5 - x^3 - x^2 - x - 1)$.
- One of the common error detecting codes is the Cyclic Redundancy Check (CRC) codes. For a k-bit block of bits the (n, k) CRC encoder generates $(n - k)$ bit long **Frame Check Sequence** (FCS).
- Shift registers can be used to encode and decode cyclic codes easily. Encoding and decoding of cyclic codes require multiplication and division by polynomials. The shift property of shift registers are ideally suited for such operations.

> *Everything should be made as simple as possible, but not simpler.*
>
> *Einstein, Albert (1879-1955)*

PROBLEMS

4.1 Which of the following codes are (a) cyclic, (b) equivalent to a cyclic code?
(a) {0000, 0110, 1100, 0011, 1001} over $GF(2)$.
(b) {00000, 10110, 01101, 11011} over $GF(2)$.
(c) {00000, 10110, 01101, 11011} over $GF(3)$.
(d) {0000, 1122, 2211} over $GF(3)$.
(e) The q-ary repetition code of length n.

4.2 Construct the addition and multiplication table for
(a) $F[x]/(x^2 + 1)$ defined over $GF(2)$.
(b) $F[x]/(x^2 + 1)$ defined over $GF(3)$.
Which of the above is a field?

4.3 List out all the irreducible polynomials over
(a) $GF(2)$ of degrees 1 to 5.
(b) $GF(3)$ of degrees 1 to 3.

4.4 Find all the cyclic binary codes of blocklength 5. Find the minimum distance of each code.

4.5 Suppose $x^n - 1$ is a product of r distinct irreducible polynomials over $GF(q)$. How many cyclic codes of blocklength n over $GF(q)$ exist? Comment on the minimum distance of these codes.

4.6 (a) Factorize $x^8 - 1$ over $GF(3)$.
(b) How many ternary cyclic codes of length 8 exist ?
(c) How many quaternary cyclic codes of length 8 exist ?

4.7 Let the polynomial
$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$
be the generator polynomial of a cyclic code over $GF(2)$ with blocklength 15.
(a) Find the generator matrix $\boldsymbol{G}$.
(b) Find the parity check matrix $\boldsymbol{H}$.
(c) How many errors can this code detect?
(d) How many errors can this code correct?
(e) Write the generator matrix in the systematic form.

4.8 Consider the polynomial
$g(x) = x^6 + 3x^5 + x^4 + x^3 + 2x^2 + 2x + 1$.
(a) Is this a valid generator polynomial for a cyclic code over $GF(4)$ with blocklength 15?
(b) Find the parity check matrix $\boldsymbol{H}$.
(c) What is the minimum distance of this code?
(d) What is the code rate of this code?
(e) Is the received word, $v(x) = x^8 + x^5 + 3x^4 + x^3 + 3x + 1$, a valid codeword?

4.9 An error vector of the form $x^i + x^{i+1}$ in R_n is called a double adjacent error. Show that the code generated by the generator polynomial $g_1(x) = (x - 1)\, g_H(x)$ is capable of correcting all double adjacent errors, where $g_H(x)$ is, the generator polynomial of the binary Hamming Code.

4.10 Design the Shift Register encoder and the Meggitt Decoder for the code generated in Problem 4.8.

4.11 The code with the generator polynomial $g(x) = (x^{23} + 1)(x^{17} + x^3 + 1)$ is used for error-detection and correction in the GSM standard.
(i) How many *random* errors can this code correct?
(ii) How many *burst* errors can this code correct?

COMPUTER PROBLEMS

4.12 Write a computer program to find the minimum distance of a cyclic code over $GF(q)$, given the generator polynomial (or the generator matrix) for the code.

4.13 Write a computer program to encode and decode a (35, 27) Fire Code. It should be able to automatically correct bursts of length 3 or less. What happens when you try to decode a received word with a burst error of length 4?

5

Bose–Chaudhuri Hocquenghem (BCH) Codes

Although this may seem a paradox, all exact science is dominated by the idea of approximation.

Russell, Bertrand (1872-1970)

5.1 INTRODUCTION TO BCH CODES

The class of Bose–Chaudhuri Hocquenghem (BCH) codes is one of the most powerful known class of Linear Cyclic Block Codes. BCH codes are known for their multiple error correcting ability, and the ease of encoding and decoding. So far, our approach has been to construct a code and then find out its minimum distance in order to estimate its error correcting capability. In this class of code, we will start from the other end. We begin by specifying the number of random errors we desire the code to correct. Then we go on to construct the generator polynomial for the code. As mentioned above, BCH codes are a subclass of cyclic codes, and therefore, the decoding methodology for any cyclic code also works for the BCH codes. However, more efficient decoding procedures are known for BCH codes, and will be discussed in this chapter.

We begin by building the necessary mathematical tools in the next couple of sections. We shall then look at the method for constructing the generator polynomial for BCH codes. Efficient decoding techniques for this class of codes will be discussed next. An important sub-set of BCH codes, the Reed-Solomon codes, will be introduced in the later part of this chapter.

5.2 PRIMITIVE ELEMENTS

Definition 5.1 A **Primitive Element** of $GF(q)$ is an element α such that every field element except zero can be expressed as a power of α.

Example 5.1 Consider $GF(5)$. Since $q = 5$ = a prime number, modulo arithmetic will work. Consider the element 2.

$$2^0 = 1 \pmod 5 = 1,$$
$$2^1 = 2 \pmod 5 = 2,$$
$$2^2 = 4 \pmod 5 = 4,$$
$$2^3 = 8 \pmod 5 = 3.$$

Hence, all the elements of $GF(5)$, i.e., $\{1, 2, 3, 4, 5\}$ can be represented as powers of 2. Therefore, 2 is a primitive element of $GF(5)$.

Next, consider the element 3.

$$3^0 = 1 \pmod 5 = 1,$$
$$3^1 = 3 \pmod 5 = 3,$$
$$3^2 = 9 \pmod 5 = 4,$$
$$3^3 = 27 \pmod 5 = 2.$$

Again, all the elements of $GF(5)$, i.e., $\{1, 2, 3, 4, 5\}$ can be represented as powers of 3. Therefore, 3 is also a primitive element of $GF(5)$.

However, it can be tested that the other non-zero elements $\{1, 4, 5\}$ are not primitive elements.

We saw in the example that there can be more than one primitive element in a field. But is there a guarantee of finding at least one primitive element? The answer is yes! The non-zero elements of every Galois Field form a cyclic group. Hence, a Galois Field will include an element of order $q - 1$. This will be the primitive element. Primitive elements are very useful in constructing fields. Once we have a primitive element, we can easily find all the other elements by simply evaluating the powers of the primitive element.

Definition 5.2 A **Primitive Polynomial** $p(x)$ over $GF(q)$ is a prime polynomial over $GF(q)$ with the property that in the extension field constructed modulo $p(x)$, the field element represented by x is a primitive element.

Primitive polynomials of every degree exist over every Galois Field. A primitive polynomial can be used to construct an extension field.

Example 5.2 **We can construct $GF(8)$ using the primitive polynomial $p(x) = x^3 + x + 1$. Let the primitive element of $GF(8)$ be $\alpha = z$. Then, we can represent all the elements of $GF(8)$ by the powers of α evaluated modulo $p(x)$. Thus, we can form Table 5.1.**

Table 5.1 *The elements of GF(8).*

Powers of α	Elements of GF(8)
α^1	z
α^2	z^2
α^3	$z+1$
α^4	z^2+z
α^5	z^2+z+1
α^6	z^2+1
α^7	1

Theorem 5.1 Let $\beta_1, \beta_2, \ldots, \beta_{q-1}$ denote the non-zero field elements of $GF(q)$. Then,

$$x^{q-1} - 1 = (x - \beta_1)(x - \beta_2) \ldots (x - \beta_{q-1}). \tag{5.1}$$

Proof The set of non-zero elements of $GF(q)$ is a finite group under the operation of multiplication. Let β be any non-zero element of the field. It can be represented as a power of the primitive element α. Let $\beta = (\alpha)^r$ for some integer r. Therefore,

$$\beta^{q-1} = ((\alpha)^r)^{q-1} = ((\alpha)^{q-1})^r = (1)^r = 1$$

because,

$$(\alpha)^{q-1} = 1.$$

Hence,

$$\beta \text{ is a zero of } x^{q-1} - 1.$$

This is true for *any* non-zero element β.

Hence,

$$x^{q-1} - 1 = (x - \beta_1)(x - \beta_2) \ldots (x - \beta_{q-1}).$$

Example 5.3 **Consider the field $GF(5)$. The non-zero elements of this field are $\{1, 2, 3, 4\}$. Therefore, we can write**

$$x^4 - 1 = (x - 1)(x - 2)(x - 3)(x - 4).$$

5.3 MINIMAL POLYNOMIALS

In the previous chapter we saw that in order to find the generator polynomials for cyclic codes of blocklength n, we have to first factorize $x^n - 1$. Thus $x^n - 1$ can be written as the product of its p prime factors

$$x^n - 1 = f_1(x)\, f_2(x)\, f_3(x) \ldots f_p(x). \tag{5.2}$$

Any combination of these factors can be multiplied together for a generator polynomial $g(x)$. If the prime factors of $x^n - 1$ are distinct, then there are $(2^p - 2)$ different non-trivial cyclic codes of blocklength n. The two trivial cases that are being disregarded are $g(x) = 1$ and $g(x) = x^n - 1$. Not all of the $(2^p - 2)$ possible cyclic codes are good codes in terms of their minimum distance. We now evolve a strategy for finding good codes, i.e., of desirable minimum distance.

In the previous chapter we learnt how to construct an extension field from the subfield. In this section we will study the prime polynomials (in a certain field) that have zeros in the extension field. Our strategy for constructing $g(x)$ will be as follows. Using the desirable zeros in the extension field we will find prime polynomials in the subfield, which will be multiplied together to yield a desirable $g(x)$.

Definition 5.3 A blocklength n of the form $n = q^m - 1$ is called a **Primitive Block Length** for a code over $GF(q)$. A cyclic code over $GF(q)$ of primitive blocklength is called a **Primitive Cyclic Code**.

The field $GF(q^m)$ is an extension field of $GF(q)$. Let the primitive block length $n = q^m - 1$. Consider the factorization

$$x^n - 1 = x^{q^m - 1} - 1 = f_1(x)\, f_2(x) \ldots f_p(x) \tag{5.3}$$

over the field $GF(q)$. This factorization will also be valid over the extension field $GF(q^m)$ because the addition and multiplication tables of the subfield forms a part of the tables of the extension field. We also know that $g(x)$ divides $x^n - 1$, i.e., $x^{q^m - 1} - 1$, hence $g(x)$ must be the product of some of these polynomials $f_i(x)$. Also, every non-zero element of $GF(q^m)$ is a zero of $x^{q^m - 1} - 1$. Hence, it is possible to factor $x^{q^m - 1} - 1$ in the extension field $GF(q^m)$ to get

$$x^{q^m - 1} - 1 = \prod_j (x - \beta_j), \tag{5.4}$$

where β_j ranges over all the non-zero elements of $GF(q^m)$. This implies that each of the polynomials $f_i(x)$ can be represented in $GF(q^m)$ as a product of some of the linear terms, and each β_j is a zero of *exactly one* of the $f_i(x)$. This $f_i(x)$ is called the **Minimal Polynomial** of β_j.

Definition 5.4 The smallest degree polynomial with coefficients in the base field $GF(q)$ that has a zero in the extension field $GF(q^m)$ is called the **Minimal Polynomial** of β_j.

Example 5.4 Consider the subfield $GF(2)$ and its extension field $GF(8)$. Here $q = 2$ and $m = 3$. The factorization of $x^{q^m-1} - 1$ (in the subfield/extension field) yields

$$x^{q^m-1} - 1 = x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Next, consider the elements of the extension field $GF(8)$. The elements can be represented as 0, $1, z, z + 1, z^2, z^2 + 1, z^2 + z, z^2 + z + 1$ (from Example 4.10 of Chapter 4).

Therefore, we can write

$$x^{q^m-1} - 1 = x^7 - 1 = (x-1)(x-z)(x-z-1)(x-z^2)(x-z^2-1)(x-z^2-z)(x-z^2-z-1)$$
$$= (x-1)\cdot[(x-z)(x-z^2)(x-z^2-z)]\cdot[(x-z-1)(x-z^2-1)(x-z^2-z-1)].$$

It can be seen that over $GF(8)$,

$$(x^3 + x + 1) = (x-z)(x-z^2)(x-z^2-z), \text{ and}$$
$$(x^3 + x^2 + 1) = (x-z-1)(x-z^2-1)(x-z^2-z-1).$$

The multiplication and addition are carried out over $GF(8)$. Interestingly, after a little bit of algebra it is found that the coefficients of the minimal polynomial belong to $GF(2)$ only. We can now make Table 5.2.

Table 5.2 *The Elements of GF(8) in Terms of the Powers of the Primitive Element, α*

Minimal polynomial $f_i(x)$	Corresponding Elements β_j in GF(8)	Elements in Terms of powers of α
$(x-1)$	1	α^0
(x^3+x+1)	z, z^2 and z^2+z	$\alpha^1, \alpha^2, \alpha^4$
(x^3+x^2+1)	$z+1$, z^2+1 and z^2+z+1	$\alpha^3, \alpha^6, \alpha^5\ (=\alpha^{12})$

It is interesting to note the elements (in terms of powers of the primitive element α) that correspond to the same minimal polynomial. If we make the observation that $\alpha^{12} = \alpha^7 \cdot \alpha^5 = 1 \cdot \alpha^5$, we see a pattern in the elements that correspond to a certain minimal polynomial. In fact, the elements that are roots of a minimal polynomial in the extension field are of the type $\beta^{q^{r-1}}$ where β is an element of the extension field. In the above example, the zeros of the minimal polynomial $f_2(x) = x^3 + x + 1$ are α^1, α^2 and α^4 and that of $f_3(x) = x^3 + x^2 + 1$ are α^3, α^6 and α^{12}.

Definition 5.5 Two elements of $GF(q^m)$ that share the same minimal polynomial over $GF(q)$ are called **Conjugates** with respect to $GF(q)$.

Example 5.5 The elements $\{\alpha^1, \alpha^2, \alpha^4\}$ are conjugates with respect to $GF(2)$. They share the same minimal polynomial $f_2(x) = x^3 + x + 1$.

As we have seen, a single element in the extension field may have more than one conjugate. The conjugacy relationship between two elements depends on the base field. For example, the extension field $GF(16)$ can be constructed using either $GF(2)$ or $GF(4)$. Two elements that are conjugates of $GF(2)$ may not be conjugates of $GF(4)$.

If $f(x)$ is the minimal polynomial of β, then it is also the minimal polynomial of the elements in the set $\{\beta, \beta^q, \beta^{q^2}, ..., \beta^{q^{r-1}}\}$, where r is the smallest integer such that $\beta^{q^{r-1}} = \beta$. The set $\{\beta, \beta^q, \beta^{q^2}, ..., \beta^{q^{r-1}}\}$ is called the **Set of Conjugates**. The elements in the set of conjugates are all the zeros of $f(x)$. Hence, the minimal polynomial of β can be written as

$$f(x) = (x-\beta)(x-\beta^q)(x-\beta^{q^2})\ldots(x-\beta^{q^{r-1}}). \tag{5.5}$$

Example 5.6 Consider $GF(256)$ as an extension field of $GF(2)$. Let α be the primitive element of $GF(256)$. Then a set of conjugates would be

$$\{\alpha^1, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}, \alpha^{128}\}$$

Note that $\alpha^{256} = \alpha^{255}\,\alpha^1 = \alpha^1$, hence the set of conjugates terminates with α^{128}. The minimal polynomial of α is

$$f(x) = (x-\alpha^1)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)(x-\alpha^{16})(x-\alpha^{32})(x-\alpha^{64})(x-\alpha^{128})$$

The right hand side of the equation when multiplied out would only contain coefficients from $GF(2)$.

Similarly, the minimal polynomial of α^3 would be

$$f(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^{24})(x-\alpha^{48})(x-\alpha^{96})(x-\alpha^{192})(x-\alpha^{129})$$

Definition 5.6 BCH codes defined over $GF(q)$ with blocklength $q^m - 1$ are called **Primitive BCH codes.**

Having developed the necessary mathematical tools, we shall now begin our study of BCH codes. We will develop a method for constructing the generator polynomials of BCH codes that can correct pre-specified t random errors.

5.4 GENERATOR POLYNOMIALS IN TERMS OF MINIMAL POLYNOMIALS

We know that $g(x)$ is a factor of $x^n - 1$. Therefore, the generator polynomial of a cyclic code can be written in the form

$$g(x) = \text{LCM}\,[f_1(x)\, f_2(x), ..., f_p(x)], \tag{5.6}$$

where, $f_1(x)\, f_2(x), ..., f_p(x)$ are the minimal polynomials of the zeros of $g(x)$. Each minimal polynomial corresponds to a zero of $g(x)$ in an extension field. We will design good codes (i.e., determine the generator polynomials) with desirable zeros using this approach.

Let $c(x)$ be a codeword polynomial and $e(x)$ be an error polynomial. Then the received polynomial can be written as

$$v(x) = c(x) + e(x) \tag{5.7}$$

where the polynomial coefficients are in $GF(q)$. Now consider the extension field $GF(q^m)$. Let γ_1, γ_2, ... γ_p be those elements of $GF(q^m)$ which are the zeros of $g(x)$, i.e., $g(\gamma_i) = 0$ for $i = 1, .., p$. Since, $c(x) = a(x)g(x)$ for some polynomial $a(x)$, we also have $c(\gamma_i) = 0$ for $i = 1, .., p$. Thus,

$$\begin{aligned} v(\gamma_i) &= c(\gamma_i) + e(\gamma_i) \\ &= e(\gamma_i) \text{ for } i = 1, ..., p \end{aligned} \tag{5.8}$$

For a blocklength n, we have

$$v(\gamma_i) = \sum_{j=0}^{n-1} e_j \gamma_i^j \text{ for } i = 1, ..., p. \tag{5.9}$$

Thus, we have a set of p equations that involve components of the error pattern only. If it is possible to solve this set of equations for e_j, the error pattern can be precisely determined. Whether this set of equations can be solved depends on the value of p, the number of zeros of $g(x)$. In order to solve for the error pattern, we must choose the set of p equations properly. If we have to design for a t error correcting cyclic code, our choice should be such that the set of equations can solve for at most t non-zero e_j.

Let us define the syndromes $S_i = e(\gamma_i)$ for $i = 1, ..., p$. We wish to choose γ_1, γ_2,... γ_p in such a manner that t errors can be computed from S_1, S_1,..., S_p. If α is a primitive element, then the set of γ_i which allow the correction of t errors is $\{\alpha^1, \alpha^2, \alpha^3, ..., \alpha^{2t}\}$. Thus, we have a simple mechanism of determining the generator polynomial of a BCH code that can correct t errors.

Steps for Determining the Generator Polynomial of a t-error Correcting BCH Code:

For a primitive blocklength $n = q^m - 1$:

(i) Choose a prime polynomial of degree m and construct $GF(q^m)$.

(ii) Find $f_i(x)$, the minimal polynomial of α^i for $i = 1, ..., p$.

(iii) The generator polynomial for the t error correcting code is simply

$$g(x) = \text{LCM}\ [f_1(x)\, f_2(x), ..., f_{2t}(x)]. \tag{5.10}$$

Codes designed in this manner can correct at least t errors. In many cases the codes will be able to correct more than t errors. For this reason,

$$d = 2t + 1 \tag{5.11}$$

is called the **Designed Distance** of the code, and the minimum distance $d^* \geq 2t + 1$. The generator polynomial has a degree equal to $n - k$ (see **Theorem 4.4**, Chapter 4). It should be noted that once we fix n and t, we can determine the generator polynomial for the BCH code. The information length k is, then, decided by the degree of $g(x)$. Intuitively, for a fixed blocklength n, a larger value of t will force the information length k to be smaller (because a higher redundancy will be required to correct more number of errors). In the following section, we look at a few specific examples of BCH codes.

5.5 SOME EXAMPLES OF BCH CODES

The following example illustrates the construction of the extension field $GF(16)$ from $GF(2)$. The minimal polynomials obtained will be used in the subsequent examples.

Example 5.7 **Consider the primitive polynomial $p(z) = z^4 + z + 1$ over $GF(2)$. We shall use this to construct the extension field $GF(16)$. Let $\alpha = z$ be the primitive element. The elements of $GF(16)$ as powers of α and the corresponding minimal polynomials are listed in the Table 5.3.**

Table 5.3 *The elements of GF(16) and the corresponding minimal polynomials*

Powers of α	Elements of GF(16)	Minimal Polynomials
α^1	z	$x^4 + x + 1$
α^2	z^2	$x^4 + x + 1$
α^3	z^3	$x^4 + x^3 + x^2 + x + 1$
α^4	$z + 1$	$x^4 + x + 1$
α^5	$z^2 + z$	$x^2 + x + 1$
α^6	$z^3 + z^2$	$x^4 + x^3 + x^2 + x + 1$
α^7	$z^3 + z + 1$	$x^4 + x^3 + 1$
α^8	$z^2 + 1$	$x^4 + x + 1$
α^9	$z^3 + z$	$x^4 + x^3 + x^2 + x + 1$
α^{10}	$z^2 + z + 1$	$x^2 + x + 1$
α^{11}	$z^3 + z^2 + z$	$x^4 + x^3 + 1$
α^{12}	$z^3 + z^2 + z + 1$	$x^4 + x^3 + x^2 + x + 1$
α^{13}	$z^3 + z^2 + 1$	$x^4 + x^3 + 1$
α^{14}	$z^3 + 1$	$x^4 + x^3 + 1$
α^{15}	1	$x + 1$

Example 5.8 We wish to determine the generator polynomial of a *single* error correcting BCH code, i.e., $t = 1$ with a blocklength $n = 15$. From (5.10), the generator polynomial for a BCH code is given by LCM $[f_1(x)\, f_2(x), ..., f_{2t}(x)]$. We will make use of Table 5.3 to obtain the minimal polynomials $f_1(x)$ and $f_2(x)$. Thus, the generator polynomial of the single error correcting BCH code will be

$$\begin{aligned} g(x) &= \text{LCM}\ [f_1(x), f_2(x)] \\ &= \text{LCM}\ [(x^4 + x + 1), (x^4 + x + 1)] \\ &= x^4 + x + 1 \end{aligned}$$

Since, deg $(g(x)) = n - k$, we have $n - k = 4$, which gives $k = 11$. Thus we have obtained the generator polynomial of the BCH (15, 11) single error correcting code. The designed distance of this code $d = 2t + 1 = 3$. It can be calculated that the minimum distance d^* of this code is also 3. Thus, in this case the designed distance is equal to the minimum distance.

Next, we wish to determine the generator polynomial of a *double* error correcting BCH code, i.e., $t = 2$ with a blocklength $n = 15$. The generator polynomial of the BCH code will be

$$g(x) = \text{LCM } [f_1(x), f_2(x), f_3(x), f_4(x)]$$
$$= \text{LCM } [(x^4 + x + 1)(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^4 + x + 1)]$$
$$= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)$$
$$= x^8 + x^7 + x^6 + x^4 + 1$$

Since, deg $(g(x)) = n - k$, we have $n - k = 8$, which gives $k = 7$. Thus, we have obtained the generator polynomial of the BCH (15, 7) double error correcting code. The designed distance of this code $d = 2t + 1 = 5$. It can be calculated that the minimum distance d^* of this code is also 5. Thus, in this case the designed distance is equal to the minimum distance.

Next, we determine the generator polynomial for the *triple* error correcting binary BCH code. The generator polynomial of the BCH code will be

$$g(x) = \text{LCM } [f_1(x) f_2(x), f_3(x), f_4(x), f_5(x), f_6(x)]$$
$$= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$$
$$= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

In this case, deg $(g(x)) = n - k = 10$, which gives $k = 5$. Thus we have obtained the generator polynomial of the BCH (15, 5) triple error correcting code. The designed distance of this code $d = 2t + 1 = 7$. It can be calculated that the minimum distance d^* of this code is also 7. Thus in this case the designed distance is equal to the minimum distance.

Next, we determine the generator polynomial for a binary BCH code for the case $t = 4$. The generator polynomial of the BCH code will be

$$g(x) = \text{LCM } [f_1(x) f_2(x), f_3(x), f_4(x), f_5(x), f_6(x) f_7(x), f_8(x)]$$
$$= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1)$$
$$= x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$

In this case, deg $(g(x)) = n - k = 14$, which gives $k = 1$. It can be seen that this is the simple repetition code. The designed distance of this code $d = 2t + 1 = 9$. However, it can be seen that the minimum distance d^* of this code is 15. Thus in this case the designed distance is not equal to the minimum distance, and the code is over designed. This code can actually correct $(d^* - 1)/2 = 7$ random errors!

If we repeat the exercise for $t = 5$, 6 or 7, we get the same generator polynomial (repetition code). Note that there are only 15 non-zero field elements in $GF(16)$ and hence there are only 15 minimal polynomials corresponding to these field elements. Thus, we cannot go beyond $t = 7$ (because for $t = 8$ we need $f_{16}(x)$, which is undefined). Hence, to obtain BCH codes that can correct larger number of errors we must use an extension field with more elements!

Example 5.9 We can construct $GF(16)$ as an extension field of $GF(4)$ using the primitive polynomial $p(z) = z^2 + z + 1$ over $GF(4)$. Let the elements of $GF(4)$ consist of the quaternary symbols contained in the set {0, 1, 2, 3}. The addition and multiplication tables for $GF(4)$ are given below for handy reference.

***GF*(4)**

+	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

.	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Table 5.4 lists the elements of $GF(16)$ as powers of α and the corresponding minimal polynomials.

Table 5.4

Powers of α	*Elements of GF(16)*	*Minimal Polynomials*
α^1	z	$x^2 + x + 2$
α^2	$z + 2$	$x^2 + x + 3$
α^3	$3z + 2$	$x^2 + 3x + 1$
α^4	$z + 1$	$x^2 + x + 2$
α^5	2	$x + 2$
α^6	$2z$	$x^2 + 2x + 1$
α^7	$2z + 3$	$x^2 + 2x + 2$
α^8	$z + 3$	$x^2 + x + 3$
α^9	$2z + 2$	$x^2 + 2x + 1$
α^{10}	3	$x + 3$
α^{11}	$3z$	$x^2 + 3x + 3$
α^{12}	$3z + 1$	$x^2 + 3x + 1$
α^{13}	$2z + 1$	$x^2 + 2x + 2$
α^{14}	$3z + 3$	$x^2 + 3x + 3$
α^{15}	1	$x + 1$

For $t = 1$,

$$g(x) = \text{LCM } [f_1(x), f_2(x)]$$
$$= \text{LCM } [(x^2 + x + 2)(x^2 + x + 3)]$$
$$= x^4 + x + 1$$

Since, deg $(g(x)) = n - k$, we have $n - k = 4$, which gives $k = 11$. Thus we have obtained the generator polynomial of the *single* error correcting BCH (15, 11) code over $GF(4)$. It takes in 11 quaternary information symbols and encodes them into 15 quaternary symbols. Note that one quaternary symbol is equivalent to two bits. So, in effect, the BCH (15, 11) takes in 22 input bits and transforms them into 30 encoded bits (can this code be used to correct a burst of length 2 for a binary sequence of length 30?). The designed distance of this code $d = 2t + 1 = 3$. It can be calculated that the minimum distance d^* of this code is also 3. Thus in this case the designed distance is equal to the minimum distance.

For $t = 2$,

$$g(x) = \text{LCM } [f_1(x), f_2(x), f_3(x), f_4(x)]$$
$$= \text{LCM } [(x^2 + x + 2), (x^2 + x + 3), (x^2 + 3x + 1), (x^2 + x + 2)]$$
$$= (x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1)$$
$$= x^6 + 3x^5 + x^4 + x^3 + 2x^2 + 2x + 1.$$

This is the generator polynomial of a (15, 9) *double* error correcting BCH code over $GF(4)$.

For $t = 3$,

$$g(x) = \text{LCM } [f_1(x), f_2(x), f_3(x), f_4(x), f_5(x), f_6(x)]$$
$$= \text{LCM } [(x^2 + x + 2), (x^2 + x + 3), (x^2 + 3x + 1), (x^2 + x + 2), (x + 2), (x^2 + 2x + 1)]$$
$$= (x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1)(x + 2)(x^2 + 2x + 1)$$
$$= x^9 + 3x^8 + 3x^7 + 2x^6 + x^5 + 2x^4 + x + 2$$

This is the generator polynomial of a (15, 6) *triple* error correcting BCH code over $GF(4)$.

Similarly, for $t = 4$,

$$g(x) = x^{11} + x^{10} + 2x^8 + 3x^7 + 3x^6 + x^5 + 3x^4 + x^3 + x + 3.$$

This is the generator polynomial of a (15, 4) *four* error correcting BCH code over $GF(4)$.

Similarly, for $t = 5$,

$$g(x) = x^{12} + 2x^{11} + 3x^{10} + 2x^9 + 2x^8 + x^7 + 3x^6 + 3x^4 + 3x^3 + x^2 + 2.$$

This is the generator polynomial of a (15, 3) *five* error correcting BCH code over $GF(4)$.

Similarly, for $t = 6$,

$$g(x) = x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.$$

This is the generator polynomial of a (15, 1) *six* error correcting BCH code over $GF(4)$. As is obvious, this is the simple repetition code, and can correct up to 7 errors.

Table 5.5 lists the generator polynomial of binary BCH codes of length up to $2^5 - 1$. Suppose we wish to construct generator polynomial of the BCH(15,7) code. From the table we have (111 010 001) for the coefficients of the generator polynomial. Therefore,

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1$$

Table 5.5 *The generator polynomials of binary BCH codes of length up to $2^5 - 1$*

n	k	t	Generator Polynomial Coefficients
7	4	1	1 011
15	11	1	10 011
15	7	2	111 010 001
15	5	3	10 100 110 111
31	26	1	100 101

Contd.

n	k	t	Generator Polynomial Coefficients
31	21	2	11 101 101 001
31	16	3	1 000 111 110 101 111
31	11	5	101 100 010 011 011 010 101
31	6	7	11 001 011 011 110 101 000 100 111

5.6 DECODING OF BCH CODES

So far we have learnt to obtain the generator polynomial for a BCH code given the number of random errors to be corrected. With the knowledge of the generator polynomial, very fast encoders can be built in hardware. We now shift our attention to the decoding of the BCH codes. Since the BCH codes are a subclass of the cyclic codes, any standard decoding procedure for cyclic codes is also applicable to BCH codes. However, better, more efficient algorithms have been designed specifically for BCH codes. We discuss the **Gorenstein-Zierler** decoding algorithm, which is the generalized form of the binary decoding algorithm first proposed by Petersen.

We develop here the decoding algorithm for a t error correcting BCH code. Suppose a BCH code is constructed based on the field element α. Consider the error polynomial

$$e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \ldots + e_1x + e_0 \tag{5.12}$$

where at most t coefficients are non-zero. Suppose that v errors actually occur, where $0 \le v \le t$. Let these errors occur at locations $i_1, i_2, \ldots, i_v$. The error polynomial can then be written as

$$e(x) = e_{i_1} x^{i_1} + e_{i_2} x^{i_2} + \ldots + e_{i_v} x^{i_v} \tag{5.13}$$

where e_{i_k} is the magnitude of the kth error. Note that we are considering the general case. For binary codes, $e_{i_k} = 1$. For error correction, we must know two things:

(i) *where* the errors have occurred, i.e., the error locations, and

(ii) what the *magnitudes* of these errors are.

Thus, the unknowns are $i_1, i_2, \ldots, i_v$ and $e_{i_1}, e_{i_2}, \ldots, e_{i_v}$, which signify the locations and the magnitudes of the errors respectively. The syndrome can be obtained by evaluating the received polynomial at α.

$$S_1 = v(\alpha) = c(\alpha) + e(\alpha) = e(\alpha)$$
$$= e_{i_1} x^{i_1} + e_{i_2} x^{i_2} + \ldots + e_{i_v} x^{i_v} \tag{5.14}$$

Next, define the error magnitudes, $Y_k = e_{i_k}$ for $k = 1, 2, \ldots, v$ and the error locations $X_k = \alpha^{i_k}$ for $k = 1, 2, \ldots, v$, where i_k is the location of the kth error and X_k is the field element associated with this location. Now, the syndrome can be written as

$$S_1 = Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \tag{5.15}$$

We can evaluate the received polynomial at each of the powers of α that has been used to define $g(x)$. We define the syndromes for $j = 1, 2, \ldots, 2t$ by

$$S_j = v(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) \tag{5.16}$$

Thus, we have the following set of $2t$ simultaneous equations, with v unknown error locations $X_1, X_2, \ldots, X_v$ and the v unknown error magnitudes $Y_1, Y_2, \ldots, Y_v$.

$$\begin{aligned} S_1 &= Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \\ S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2 \\ &\vdots \\ S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t} \end{aligned} \tag{5.17}$$

Next, define the error locator polynomial

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1} x^{v-1} + \ldots \Lambda_1 x + 1 \tag{5.18}$$

The zeros of this polynomial are the inverse error locations X_k^{-1} for $k = 1, 2, \ldots, v$. That is,

$$\Lambda(x) = (1 - xX_1)(1 - xX_2) \ldots (1 - xX_v) \tag{5.19}$$

So, if we know the coefficients of the error locator polynomial $\Lambda(x)$, we can obtain the error locations $X_1, X_2, \ldots, X_v$. After some algebraic manipulations we obtain

$$\Lambda_1 S_{j+v-1} + \Lambda_2 S_{j+v-2} + \ldots + \Lambda_v S_{j+v} \text{ for } j = 1, 2, \ldots, v \tag{5.20}$$

This is nothing but a set of linear equations that relate the syndromes to the coefficients of $\Lambda(x)$. This set of equations can be written in the matrix form as follows.

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{v-1} & S_v \\ S_2 & S_3 & \cdots & S_v & S_{v+1} \\ S_v & S_{v+1} & \cdots & S_{2v-2} & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ -S_{2v} \end{bmatrix} \tag{5.21}$$

The values of the coefficients of the error locator polynomial can be determined by inverting the syndrome matrix. This is possible only if the matrix is non-singular. It can be shown that this matrix is non-singular if there are v errors.

Steps for Decoding BCH Codes

(i) As a trial value, set $v = t$ and compute the determinant of the matrix of syndromes, $\boldsymbol{M}$. If the determinant is zero, set $v = t - 1$. Again compute the determinant of $\boldsymbol{M}$. Repeat this process until a value of v is found for which the determinant of the matrix of syndromes is non-zero. This value of v is the actual number of errors that occurred.

(ii) Invert the matrix $\boldsymbol{M}$ and find the coefficients of the error locator polynomial $\Lambda(x)$.

(iii) Solve $\Lambda(x) = 0$ to obtain the zeros and from them compute the error locations $X_1, X_2, \ldots, X_v$. If it is a binary code, stop (because the magnitudes of error are unity).

(iv) If the code is not binary, go back to the system of equations:

$$\begin{aligned} S_1 &= Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \\ S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2 \\ &\vdots \\ S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t} \end{aligned}$$

Since the error locations are now known, these form a set of $2t$ *linear* equations. These can be solved to obtain the error magnitudes.

Solving for Λ_i by inverting the $v \times v$ matrix can be computationally expensive. The number of computations required will be proportional to v^3. If we need to correct a large number of errors (i.e., a large v) we need more efficient ways to solve the matrix equation. Various refinements have been found which greatly reduce the computational complexity. It can be seen that the $v \times v$ matrix is not arbitrary in form. The entries in its diagonal perpendicular to the main diagonal are all identical. This property is called *persymmetry*. This structure was exploited by Berleykamp (1968) and Massey (1969) to find a simpler solution to the system of equations.

The simplest way to search for the zeros of $\Lambda(x)$ is to test out all the field elements one by one. This method of exhaustive search is known as the *Chien search*.

Example 5.10 Consider the BCH (15, 5) triple error correcting code with the generator polynomial

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

Let the all zero codeword be transmitted and the received polynomial be $v(x) = x^5 + x^3$. Thus, there are two errors at the locations 4 and 6. The error polynomial $e(x) = x^5 + x^3$. But the decoder does not know this. It does not even know how many errors have actually occurred. We use the Gorenstein-Zierler Decoding Algorithm. First we compute the syndromes using the arithmetic of $GF(16)$.

$$\begin{aligned} S_1 &= \alpha^5 + \alpha^3 = \alpha^{11} \\ S_2 &= \alpha^{10} + \alpha^6 = \alpha^7 \\ S_3 &= \alpha^{15} + \alpha^9 = \alpha^7 \\ S_4 &= \alpha^{20} + \alpha^{12} = \alpha^{14} \\ S_5 &= \alpha^{25} + \alpha^{15} = \alpha^5 \\ S_6 &= \alpha^{30} + \alpha^{18} = \alpha^{14} \end{aligned}$$

First set $v = t = 3$, since this is a triple error correcting code.

$$\boldsymbol{M} = \begin{bmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^7 & \alpha^7 \\ \alpha^7 & \alpha^7 & \alpha^{14} \\ \alpha^7 & \alpha^{14} & \alpha^5 \end{bmatrix}$$

Det $(\boldsymbol{M}) = 0$, which implies that fewer than 3 errors have occurred. Next, set $v = t = 2$.

$$\boldsymbol{M} = \begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^7 \\ \alpha^7 & \alpha^7 \end{bmatrix}$$

Det $(M) \neq 0$, which implies that 2 errors have actually occurred. We next calculate M^{-1}. It so happens that in this case,

$$M^{-1} = \begin{bmatrix} \alpha^{11} & \alpha^{7} \\ \alpha^{7} & \alpha^{7} \end{bmatrix}$$

$$\begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_3 \\ -S_4 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^{7} \\ \alpha^{7} & \alpha^{7} \end{bmatrix} \begin{bmatrix} \alpha^{7} \\ \alpha^{14} \end{bmatrix}$$

Solving for Λ_1 and Λ_2 we get $\Lambda_2 = \alpha^8$ and $\Lambda_1 = \alpha^{11}$. Thus,

$$\Lambda(x) = \alpha^8 x^2 + \alpha^{11} x + 1 = (\alpha^5 x + 1)(\alpha^3 x + 1).$$

Thus, the recovered error locations are α^5 and α^3. Since the code is binary, the error magnitudes are 1. Thus, $e(x) = x^5 + x^3$.

In the next section, we will study the famous Reed–Solomon Codes, an important sub-class of BCH codes.

5.7 REED-SOLOMON CODES

Reed–Solomon (RS) Codes are an important subclass of the non-binary BCH with a wide range of applications in digital communications and data storage. The typical application areas of the RS code are

- Storage devices (including tape, Compact Disk, DVD, barcodes, etc),
- Wireless or mobile communication (including cellular telephones, microwave links, etc),
- Satellite communication,
- Digital television / Digital Video Broadcast (DVB),
- High-speed modems such as those employing ADSL, xDSL, etc.

It all began with a five-page paper that appeared in 1960 in the *Journal of the Society for Industrial and Applied Mathematics*. The paper, "Polynomial Codes over Certain Finite Fields" by Irving S. Reed and Gustave Solomon of MIT's Lincoln Laboratory, introduced the ideas that form a significant portion of current error correcting techniques for everything from computer hard disk drives to CD players. Reed–Solomon Codes (plus a lot of engineering wizardry, of course) made possible the stunning pictures of the outer planets sent back by Voyager II. They make it possible to scratch a compact disc and still enjoy the music. And in the not-too-distant future, they will enable the profit mongers of cable television to squeeze more than 500 channels into their systems.

RS coding system is based on groups of bits, such as bytes, rather than individual 0s and 1s, making it particularly good at dealing with *bursts* of errors: six consecutive bit errors, for example, can affect at most two bytes. Thus, even a double-error-correction version of a Reed–Solomon code can provide a comfortable safety factor. Current implementations of Reed–

Solomon codes in CD technology are able to cope with error bursts as long as 4000 consecutive bits.

In this sub-class of BCH codes, the symbol field $GF(q)$ and the error locator field $GF(q^m)$ are the same, i.e., $m = 1$. Thus, in this case

$$n = q^m - 1 = q - 1 \tag{5.22}$$

The minimal polynomial of any element β in the same field $GF(q)$ is

$$f_\beta(x) = x - \beta \tag{5.23}$$

Since the symbol field (sub-field) and the error locator field (extension field) are the same, all the minimal polynomials are linear. The generator polynomial for a t error correcting code will be simply

$$\begin{aligned} g(x) &= \text{LCM}\ [f_1(x)\ f_2(x), \ldots, f_{2t}(x)] \\ &= (x - \alpha)(x - \alpha^2) \ldots (x - \alpha^{2t-1})\ (x - \alpha^{2t}) \end{aligned} \tag{5.24}$$

Hence, the degree of the generator polynomial will always be $2t$. Thus, the RS code satisfies

$$n - k = 2t \tag{5.25}$$

In general, the generator polynomial of an RS code can be written as

$$g(x) = (x - \alpha^i)(x - \alpha^{i+1}) \ldots (x - \alpha^{2t+i-1})(x - \alpha^{2t+i}) \tag{5.26}$$

Example 5.11 **Consider the double error correcting RS code of blocklength 15 over GF (16). Here $t = 2$. We use here the elements of the extension field GF (16) constructed from GF (2) using the primitive polynomial $p(z) = z^4 + z + 1$. The generator polynomial can be written as**

$$\begin{aligned} g(x) &= (x - \alpha)(x - \alpha^2)\ (x - \alpha^3)\ (x - \alpha^4) \\ &= x^4 + (z^3 + z^2 + 1)\, x^3 + (z^3 + z^2)\, x^2 + z^3 x + (z^2 + z + 1) \\ &= x^4 + (\alpha^{13})\, x^3 + (\alpha^6)\, x^2 + (\alpha^3)\, x + \alpha^{10} \end{aligned}$$

Here $n - k = 4$, which implies $k = 11$. Thus, we have obtained the generator polynomial of an RS (15, 11) code over GF (16). Note that this coding procedure takes in 11 symbols (equivalent to $4 \times 11 = 44$ bits) and encodes them into 15 symbols (equivalent to 60 bits).

Theorem 5.2 A **Reed–Solomon Code** is a **Maximum Distance Separable** (MDS) **Code** and its minimum distance is $n - k + 1$.

Proof Let the designed distance of the RS code be $d = 2t + 1$. The minimum distance d^* satisfies the condition

$$d^* \geq d = 2t + 1$$

But, for an RS code, $2t = n - k$. Hence,

$$d^* \geq d = n - k.$$

But, by the Singleton Bound for any linear code,

$$d^* \le n - k$$

Thus, $d^* = n - k + 1$, and the minimum distance $d^* = d$, the designed distance of the code. Since RS codes are maximum distance separable (MDS), all of the possible code words are as far away as possible algebraically in the code space. It implies a uniform code word distribution in the code space.

Table 5.6 lists the parameters of some RS codes. Note that for a given minimum distance, in order to have a high code rate, one must work with larger Galois Fields.

Table 5.6 *Some RS code parameters*

m	$q = 2^m$	$n = q - 1$	t	k	d	$r = k/n$
2	4	3	1	1	3	0.3333
3	8	7	1	5	3	0.7143
			2	3	5	0.4286
			3	1	7	0.1429
4	16	15	1	13	3	0.8667
			2	11	5	0.7333
			3	9	7	0.6000
			4	7	9	0.4667
			5	5	11	0.3333
			6	3	13	0.2000
			7	1	15	0.0667
5	32	31	1	29	3	0.9355
			5	21	11	0.6774
			8	15	17	0.4839
8	256	255	5	245	11	0.9608
			15	225	31	0.8824
			50	155	101	0.6078

Example 5.12 A popular Reed-Solomon code is RS(255,223) with 8-bit symbols (bytes), i.e., over $GF(256)$. Each codeword contains 255 code word bytes, of which 223 bytes are data and 32 bytes are parity. For this code, $n = 255$, $k = 223$ and $n - k = 32$. Hence, $2t = 32$, or $t = 16$. Thus, the decoder can correct any 16 symbol random error in the codeword, i.e., errors in up to 16 bytes anywhere in the codeword can be corrected.

Example 5.13 Reed Solomon error correction codes have an extremely pronounced effect on the efficiency of a digital communication channel. For example, an operation running at a data rate of 1 million bytes per second will carry approximately 4000 blocks of 255 bytes each second. If 1000 random short errors (less than 17 bits in length) per second are injected into the channel, about 600 to 800 blocks per second would be corrupted, which might require retransmission of nearly all of the blocks. By applying the Reed–Solomon (255, 235) code (that corrects up to 10 errors per block of 235 information bytes and 20 parity bytes), the typical time between blocks that cannot be corrected and would require retransmission will be about 800 years. The mean time between incorrectly decoded blocks will be over 20 billion years!

5.8 IMPLEMENTATION OF REED-SOLOMON ENCODERS AND DECODERS

Hardware Implementation

A number of commercial hardware implementations exist for RS codes. Many existing systems use *off-the-shelf* integrated circuits that encode and decode Reed–Solomon codes. These ICs tend to support a certain amount of programmability, for example, RS(255, k) where $t = 1$ to 16 symbols. The recent trend has been towards VHDL or Verilog Designs (logic cores or intellectual property cores). These have a number of advantages over standard ICs. A logic core can be integrated with other VHDL or Verilog components and synthesized to an FPGA (Field Programmable Gate Array) or ASIC (Application Specific Integrated Circuit)—this enables so-called "System on Chip" designs where multiple modules can be combined in a single IC. Depending on production volumes, logic cores can often give significantly lower system costs than standard ICs. By using logic cores, a designer avoids the potential need to do a life-time buy of a Reed–Solomon IC.

Software Implementation

Until recently, software implementations in "real-time" required too much computational power for all but the simplest of Reed–Solomon codes (i.e. codes with small values of t). The major difficulty in implementing Reed–Solomon codes in software is that general purpose processors do not support Galois Field arithmetic operations. For example, to implement a Galois Field multiply in software requires a test for 0, two log table look-ups, modulo add and anti-log table look-up. However, careful design together with increases in processor performance mean that software implementation can operate at relatively high data rates. Table 5.7 gives sample benchmark figures on a 1.6 GHz Pentium PC. These data rates are for decoding only. Encoding is considerably faster since it requires less computation.

Table 5.7 *Sample benchmark figures for software decoding of some RS codes*

Code	Data Rate	t
RS(255,251)	~ 120 Mbps	2
RS(255,239)	~ 30 Mbps	8
RS(255,223)	~ 10 Mbps	16

5.9 NESTED CODES

One of the ways to achieve codes with large blocklengths is to *nest* codes. This technique combines a code of a small alphabet size and one of a larger alphabet size. Let a block of q-ary symbols be of length kK. This block can be broken up into K sub-blocks of k symbols. Each sub-block can be viewed as an element of a q^k-ary alphabet. A sequence of K such sub-blocks can be encoded with an (N, K) code over $GF(q^k)$. Now, each of the N q^k-ary symbols can be viewed as

k q-ary symbols and can be coded with an (n, k) q-ary code. Thus, a nested code has two distinct levels of coding. This method of generating a nested code is given in Fig. 5.1.

Fig. 5.1 *Nesting of Codes.*

Example 5.14 The following two codes can be nested to form a code with a larger blocklength.

Inner code: The RS (7, 3) *double* error correcting code over $GF(8)$.

Outer code: The RS (511, 505) *triple* error correcting code over $GF(8^3)$.

On nesting these codes we obtain a (3577, 1515) code over $GF(8)$. This code can correct any random pattern of 11 errors. The codeword is 3577 symbols long, where the symbols are the elements of $GF(8)$.

Example 5.15 RS codes are extensively used in the compact discs (CD) for error correction. Below we give the standard Compact Disc digital format.

Sampling frequency: 44.1 kHz, i.e., 10% margin with respect to the Nyquist frequency (audible frequencies below 20 kHz)

Quantization: 16-bit linear => theoretical SNR about 98 dB (for sinusoidal signal with maximum allowed amplitude), 2's complement

Signal format: Audio bit rate 1.41 Mbit/s (44.1 kHz × 16 bits × 2 channels), Cross Interleave Reed–Solomon Code (CIRC), total data rate (CIRC, sync, subcode) 2.034 Mbit/s.

Playing time: Maximum 74.7 min.

Disc specifications: Diameter 120 mm, thickness 1.2 mm, track pitch 1.6 μm, one side medium, disc rotates clockwise, signal is recorded from inside to outside, constant linear velocity (CLV), recording maximizes recording density (the speed of revolution of the disc is not constant; it gradually decreases from 500 to 200 r/min), pit is about 0.5 μm wide, each pit edge is '1' and all areas in between, whether inside or outside a pit, are '0's.

Error Correction: A typical error rate of a CD system is 10^{-5}, which means that a data error occurs roughly 20 times per second (bit rate × BER). About 200 error/s can be corrected.

Sources of errors: Dust, scratches, fingerprints, pit asymmetry, bubbles or defects in substrate, coating defects and dropouts.

Cross Interleave Reed–Solomon Code (CIRC)

- C2 can effectively correct burst errors.
- C1 can correct random errors and detect burst errors.
- Three interleaving stages to encode data before it is placed on a disc.
- Parity checking to correct random errors
- Cross interleaving to permit parity to correct burst errors.

1. Input stage: 12 words (16-bit, 6 words per channel) of data per input frame divided into 24 symbols of 8 bits.
2. C2 Reed Solomon code: 24 symbols of data are enclosed into a (28, 24) RS code and 4 parity symbols are used for error correction.
3. Cross interleaving: to guard against burst errors, separate error correction codes, one code can check the accuracy of another, error correction is enhanced.
4. C1 Reed–Solomon code: cross-interleaved 28 symbols of the C2 code are encoded again into a (32, 28) R–S code (4 parity symbols are used for error correction).
5. Output stage: half of the code word is subject to a 1-symbol delay to avoid 2-symbol error at the boundary of symbols.

Performance of CIRC: Both RS coders (C1 and C2) have four parities, and their minimum distance is 5. If error location is not known, up to two symbols can be corrected. If the errors exceed the correction limit, they are concealed by interpolation. Since even-numbered sampled data and odd-numbered sampled data are interleaved as much as possible, CIRC can conceal long burst errors by simple linear interpolation.

- Maximum correctable burst length is about 4000 data bits (2.5 mm track length).
- Maximum correctable burst length by interpolation in the worst case is about 12320 data bits (7.7 mm track length).

Sample interpolation rate is one sample every 10 hours at BER (Bit Error Rate) = 10^{-4} and 1000 samples at BER = 10^{-3}. Undetectable error samples (clicks) are less than one every 750 hours at BER = 10^{-3} and negligible at BER = 10^{-4}.

5.10 CONCLUDING REMARKS

The class of BCH codes were discovered independently by Hocquenghem in 1959 and Bose and Ray Chaudhuri in 1960. The BCH codes constitute one of the most important and powerful classes of linear block codes, which are cyclic.

The Reed–Solomon codes were discovered by Irving S. Reed and Gustave Solomon who published a five-page paper in the *Journal of the Society for Industrial and Applied Mathematics* in 1960 titled "Polynomial Codes over Certain Finite Fields". Despite their advantages, Reed–

Solomon codes did not go into use immediately after their invention. They had to wait for the hardware technology to catch up. In 1960, there was no such thing as fast digital electronics, at least not by today's standards. The Reed–Solomon paper suggested some nice ways to process data, but nobody knew if it was practical or not, and in 1960 it probably wasn't practical.

Eventually technology did catch up, and numerous researchers began to work on implementing the codes. One of the key individuals was Elwyn Berlekamp, a professor of electrical engineering at the University of California at Berkeley, who invented an efficient algorithm for decoding the Reed–Solomon code. Berlekamp's algorithm was used by Voyager II and is the basis for decoding in CD players. Many other bells and whistles (some of fundamental theoretic significance) have also been added. Compact discs, for example, use a version called cross-interleaved Reed–Solomon code, or CIRC.

SUMMARY

- A primitive element of $GF(q)$ is an element α such that every field element except zero can be expressed as α power of α. A field can have more than one primitive element.
- A primitive polynomial $p(x)$ over $GF(q)$ is a prime polynomial over $GF(q)$ with the property that in the extension field constructed modulo $p(x)$, the field element represented by x is a primitive element.
- A blocklength n of the form $n = q^m - 1$ is called a primitive block length for a code over $GF(q)$. A cyclic code over $GF(q)$ of primitive blocklength is called a primitive cyclic code.
- It is possible to factor $x^{q^m-1} - 1$ in the extension field $GF(q^m)$ to get $x^{q^m-1} - 1 = \prod_j (x - \beta_j)$, where β_j ranges over all the non-zero elements of $GF(q^m)$. This implies that each of the polynomials $f_i(x)$ can be represented in $GF(q^m)$ as a product of some of the linear terms, and each β_j is a zero of *exactly one* of the $f_i(x)$. This $f_i(x)$ is called the minimal polynomial of β_j.
- Two elements of $GF(q^m)$ that share the same minimal polynomial over $GF(q)$ are called conjugates with respect to $GF(q)$.
- BCH codes defined over $GF(q)$ with blocklength $q^m - 1$ are called primitive BCH codes.
- To determine the generator polynomial of a t-error correcting BCH code for a primitive blocklength $n = q^m - 1$, (i) Choose a prime polynomial of degree m and construct $GF(q^m)$, (ii) find $f_i(x)$, the minimal polynomial of α^i for $i = 1, ..., p$. (iii) obtain the generator polynomial $g(x) = \text{LCM}\ [f_1(x)\ f_2(x), ..., f_{2t}(x)]$. Codes designed in this manner can correct at least t errors. In many cases the codes will be able to correct more than t errors. For this reason, $d = 2t + 1$ is called the designed distance of the code, and the minimum distance $d^* \geq 2t + 1$.
- Steps for decoding BCH codes:
  (1) As a trial value, set $v = t$ and compute the determinant of the matrix of syndromes, $\boldsymbol{M}$. If the determinant is zero, set $v = t - 1$. Again compute the determinant of $\boldsymbol{M}$. Repeat this process until a value of v is found for which the determinant of the matrix of syndromes is non zero. This value of v is the actual number of errors that occurred.
  (2) Invert the matrix $\boldsymbol{M}$ and find the coefficients of the error locator polynomial $\Lambda(x)$.
  (3) Solve $\Lambda(x) = 0$ to obtain the zeros and from them compute the error locations $X_1, X_2, ..., X_v$. If it is a binary code, stop (because the magnitudes of error are unity).
  (4) If the code in not binary, go back to the system of equations:

$$\begin{aligned} S_1 &= Y_1X_1 + Y_2X_2 + ... + Y_vX_v \\ S_2 &= Y_1X_1^2 + Y_2X_2^2 + ... + Y_vX_v^2 \\ &\vdots \qquad\qquad\qquad \vdots \\ S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + ... + Y_vX_v^{2t} \end{aligned}$$

  Since the error locations are now known, these form a set of $2t$ *linear* equations. These can be solved to obtain the error magnitudes.
- The generator polynomial for a t error correcting RS code will be simply $g(x) = \text{LCM}[f_1(x)\ f_2(x), ..., f_{2t}(x)] = (x - \alpha)(x - \alpha^2) ... (x - \alpha^{2t-1})(x - \alpha^{2t})$. Hence, the degree of the generator polynomial will always be $2t$. Thus, the RS code satisfies $n - k = 2t$.
- A Reed–Solomon code is a Maximum Distance Separable (MDS) Code and its minimum distance is $n - k + 1$.
- One of the ways to achieve codes with large blocklengths is to *nest* codes. This technique combines a code of a small alphabet size and a code of a larger alphabet size. Let a block of q-ary symbols be of length kK. This block can be broken up into K subblocks of k symbols. Each sub-block can be viewed as an element of a q^k-ary alphabet.

> "Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth."
> - Sherlock Holmes (by Sir Arthur Conan Doyle, 1859-1930)

PROBLEMS

5.1 Construct $GF(9)$ from $GF(3)$ using an appropriate primitive polynomial.

5.2 (i) Find the generator polynomial $g(x)$ for a double error correcting *ternary* BCH code of blocklength 8. What is the code rate of this code? Compare it with the (11, 6) ternary Golay code with respect to the code rate and the minimum distance.

(ii) Next, find the generator polynomial $g(x)$ for a triple error correcting *ternary* BCH code of blocklength 26.

5.3 Find the generator polynomial $g(x)$ for a binary BCH code of blocklength 31. Use the primitive polynomial $p(x) = x^5 + x^2 + 1$ to construct $GF(32)$. What is the minimum distance of this code?

5.4 Find the generator polynomials and the minimum distance for the following codes:

(i) RS (15, 11) code

(ii) RS (15, 7) code

(iii) RS (31, 21) code.

5.5 Show that every BCH code is a subfield subcode of a Reed–Solomon Code of the same designed distance. Under what condition is the code rate of the BCH code equal to that of the RS code?

5.6 Consider the code over $GF(11)$ with a parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & 10 \\ 1 & 2^2 & 3^2 & \cdots & 10^2 \\ 1 & 2^3 & 3^3 & \cdots & 10^3 \end{bmatrix}$$

(i) Find the minimum distance of this code.

(ii) Show that this is an optimal code with respect to the Singleton Bound.

5.7 Consider the code over $GF(11)$ with a parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & 4 & \cdots & 10 \\ 1 & 2^2 & 3^2 & 4^2 & \cdots & 10^2 \\ 1 & 2^3 & 3^3 & 4^3 & \cdots & 10^3 \\ 1 & 2^4 & 3^4 & 4^4 & \cdots & 10^4 \\ 1 & 2^5 & 3^5 & 4^5 & \cdots & 10^5 \end{bmatrix}$$

(i) Show that the code is a triple error correcting code.

(ii) Find the generator polynomial for this code.

COMPUTER PROBLEMS

5.8 Write a computer program which takes in the coefficients of a primitive polynomial, the values of q and m, and then constructs the extension field $GF(q^m)$.

5.9 Write a computer program that performs addition and multiplication over $GF(2^m)$, where m is an integer.

5.10 Find the generator polynomial $g(x)$ for a binary BCH code of blocklength 63. Use the primitive polynomial $p(x) = x^6 + x + 1$ to construct $GF(64)$. What is the minimum distance of this code?

5.11 Write a program that performs BCH decoding given n, q, t and the received vector.

5.12 Write a program that outputs the generator polynomial of the Reed–Solomon code with the codeword length n and the message length k. A valid n should be $2^M - 1$, where M is an integer not less than 3. The program should also list the minimum distance of the code.

5.13 Write a computer program that performs the two level RS coding as done in a standard compact disc.

6

Convolutional Codes

The shortest path between two truths in the real domain passes through the complex domain.

Jacques Hadamard 1865-1963

6.1 INTRODUCTION TO CONVOLUTIONAL CODES

So far we have studied block codes, where a block of k information symbols are encoded into a block of n coded symbols. There is always a one-to-one correspondence between the uncoded block of symbols (information word) and the coded block of symbols (codeword). This method is particularly useful for high data rate applications, where the incoming stream of uncoded data is first broken into blocks, encoded, and then transmitted (Fig. 6.1). A large blocklength is important because of the following reasons.

(i) Many of the good codes that have large distance properties are of large blocklengths (e.g., the RS codes),

(ii) Larger blocklengths imply that the encoding overhead is small.

However, very large blocklengths have the disadvantage that unless the entire block of encoded data is received at the receiver, the decoding procedure cannot start, which may result in delays. In contrast, there is another coding scheme in which much smaller blocks of uncoded

data of length k_0 are used. These are called **Information Frames**. An information frame typically contains just a few symbols, and can have as few as just one symbol! These information frames are encoded into **Codeword Frames** of length n_0. However, just one information frame is not used to obtain the codeword frame. Instead, the current information frame with previous m information frames are used to obtain a single codeword frame. This implies that such encoders have *memory*, which retain the previous m incoming information frames. The codes that are obtained in this fashion are called **Tree Codes**. An important sub-class of Tree Codes, used frequently in practice, is called **Convolutional Codes**. Up to now, all the decoding techniques discussed are algebraic and are memoryless, i.e. decoding decisions are based only on the current codeword. Convolutional codes make decisions based on past information, i.e. memory is required.

Fig. 6.1 *Encoding Using a Block Encoder.*

In this chapter, we start with an introduction to **Tree and Trellis Codes**. We will then, develop the necessary mathematical tools to construct convolutional codes. We will see that convolutional codes can be easily represented by polynomials. Next, we will give a matrix description of convolutional codes. The chapter goes on to discuss the famous **Viterbi Decoding Technique**. We shall conclude this chapter by giving an introduction to **Turbo Coding and Decoding**.

6.2 TREE CODES AND TRELLIS CODES

We assume that we have an infinitely long stream of incoming symbols (thanks to the volumes of information sent these days, it is not a bad assumption!). This stream of symbols is first broken up into segments of k_0 symbols. Each segment is called an **Information Frame**, as mentioned earlier. The encoder consists of two parts (Fig. 6.2):

(i) memory, which basically is a shift register,

(ii) a logic circuit.

The memory of the encoder can store m information frames. Each time a new information frame arrives, it is shifted into the shift register and the oldest information frame is discarded. At the end of any frame time the encoder has m most recent information frames in its memory, which corresponds to a total of mk_0 information symbols.

When a new frame arrives, the encoder computes the codeword frame using this new frame that has just arrived and the stored previous m frames. The computation of the codeword frame is done using the logic circuit. This codeword frame is then shifted out. The oldest information frame in the memory is then discarded and the most recent information frame is shifted in. The

encoder is now ready for the next incoming information frame. Thus, for every information frame (k_0 symbols) that comes in, the encoder generates a codeword frame (n_0 symbols). It should be observed that the same information frame may not generate the same codeword frame because the codeword frame also depends on the m previous information frames.

Definition 6.1 The **Constraint Length** of a shift register encoder is defined as the number of symbols it can store in its memory. We shall give a more formal definition of constraint length later in this chapter.

If the shift register encoder stores m previous information frames of length k_0, the constraint length of this encoder $v = mk_0$.

Fig. 6.2 *A Shift Register Encoder that Generates a Tree Code.*

Definition 6.2 The infinite set of all infinitely long codewords obtained by feeding every possible input sequence to a shift register encoder is called a (k_0, n_0) **Tree Code**. The rate of this tree code is defined as

$$R = \frac{k_0}{n_0}. \tag{6.1}$$

A more formal definition is that a (k_0, n_0) **Tree Code** is a mapping from the set of semi infinite sequences of elements of $GF(q)$ into itself such that if for any m, two semi infinite sequences agree in the first mk_0 components, then their images agree in the first mk_0 components.

Definition 6.3 The **Wordlength** of a shift register encoder is defined as $k = (m + 1)k_0$. The **Blocklength** of a shift register encoder is defined as $n = (m + 1)n_0 = k\frac{n_0}{k_0}$

Note that the code rate $R = \frac{k_0}{n_0} = \frac{k}{n}$. Normally, for practical shift register encoders, the information frame length k_0 is small (usually less than 5). Therefore, it is difficult to obtain the code rate R of tree codes close to unity, as is possible with block codes (e.g., RS codes).

Definition 6.4 A (n_0, k_0) tree code that is linear, time-invariant, and has a finite wordlength $k = (m+1)k_0$ is called an (n, k) **Convolutional Code.**

Definition 6.5 A (n_0, k_0) tree code that is time-invariant and has a finite wordlength k is called an (n, k) **Sliding Block Code.** Thus, a linear sliding block code is a convolutional code.

Example 6.1 Consider the convolutional encoder given in Fig. 6.3.

Fig. 6.3 *Convolutional Encoder of Example 6.1.*

This encoder takes in one bit at a time and encodes it into 2 bits. The information frame length $k_0 = 1$, the codeword frame length $n_0 = 2$ and the blocklength $(m+1)n_0 = 6$. The constraint length of this encoder is $\nu = 2$ and the code rate ½. The clock rate of the outgoing data is twice as fast as that of incoming data. The adders are binary adders, and from the point of view of circuit implementation, are simply XOR gates.

Let us assume that the initial state of the shift register is [0 0]. Now, either '0' will come or '1' will come as the incoming bit. Suppose '0' comes. On performing the logic operations, we see that the computed value of the codeword frame is [0 0]. The 0 will be pushed into the memory (shift register) and the rightmost '0' will be dropped. The state of the shift register remains [0 0]. Next, let '1' arrive at the encoder. Again we perform the logic operations to compute the codeword frame. This time we obtain [1 1]. So, this will be pushed out as the encoded frame. The incoming '1' will be shifted into the memory, and the rightmost bit will be dropped. So the new state of the shift register will be [1 0].

Table 6.1 lists all the possibilities.

Table 6.1 *The Incoming and Outgoing Bits of the Convolutional Encoder.*

Incoming Bit	Current State of the Encoder		Outgoing Bits	
0	0	0	0	0
1	0	0	1	1
0	0	1	1	1
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0
0	1	1	1	0
1	1	1	0	1

We observe that there are only $2^2 = 4$ possible states of the shift register. So, we can construct the state diagram of the encoder as shown in Fig. 6.4. The bits associated with each arrow represents the incoming bit. It can be seen that the same incoming bit gets encoded *differently* depending on the current state of the encoder. This is different from the linear block codes studied in previous chapters where there is always a one-to-one correspondence between the incoming uncoded block of symbols (information word) and the coded block of symbols (codeword).

Fig. 6.4 *The State Diagram for the Encoder in Example 6.1.*

The same information contained in the state diagram can be conveyed usefully in terms of a graph called the **Trellis Diagram**. A trellis is a graph whose nodes are in a rectangular grid, which is semi-infinite to the right. Hence, these codes are also called **Trellis Codes**. The number of nodes in each column is finite.The following example gives an illustration of a trellis diagram.

Example 6.2 The trellis diagram for the convolutional encoder discussed in Example 6.1 is given in Fig. 6.5.

Every node in the trellis diagram represents a state of the shift register. Since the rate of the encoder is ½, one bit at a time is processed by the encoder. The incoming bit is either a '0' or a '1'. Therefore, there are two branches emanating form each node. The top branch represents the input as '0' and the lower branch corresponds to '1'. Therefore, labelling is not required for a binary trellis diagram. In general, one would label each branch with the input symbol to which it corresponds. Normally, the nodes that cannot be reached by starting at the top left node and moving only to the right are not shown in the trellis diagram. Corresponding to a certain state and a particular incoming bit, the encoder will produce an output. The output of the encoder is written on

Fig. 6.5 *The Trellis Diagram for the Encoder Given in Fig. 6.3.*

top of that branch. Thus, a trellis diagram gives a very easy method to encode a stream of input data. The encoding procedure using a trellis diagram is as follows.

- We start from the top left node (since the initial state of the encoder is [0 0]).
- Depending on whether a '0' or a '1' comes, we follow the upper or the lower branch to the next node.
- The encoder output is read out from the top of the branch being traversed.
- Again, depending on whether a '0' or a '1' comes, we follow the upper or the lower branch from the current node (state).
- Thus, the encoding procedure is simply following the branches on the diagram and reading out the encoder outputs that are written on top of each branch.

Encoding the bit stream 1 0 0 1 1 0 1 ... gives a trellis diagram as illustrated in Fig. 6.6. The encoded sequence can be read out from the diagram as 11 01 11 11 10 10 00 ... .

It can be seen that there is a one-to-one correspondence between the encoded sequence and a path in the trellis diagram. Should the decoding procedure, then, just search for the most likely path in the trellis diagram? The answer is yes, as we shall see further along in this chapter!

Fig. 6.6 *Encoding an Input Sequence Using the Trellis Diagram.*

6.3 POLYNOMIAL DESCRIPTION OF CONVOLUTIONAL CODES (ANALYTICAL REPRESENTATION)

In contrast to the two pictorial representations of the convolutional codes (the state diagram and the trellis diagram), there is a very useful analytical representation of convolutional codes. The representation makes use of the delay operator, D. We have earlier seen a one-to-one correspondence between a word (vector) and a polynomial. The delay operator is used in a similar manner. For example, consider the word 10100011 with the oldest digit at the left. The analytical representation (sometimes referred to as the *transform*) of this information word $I(D)$ will be

$$I(D) = 1 + D^2 + D^6 + D^7 \tag{6.2}$$

The **indeterminate** D is the number of time units of delay of that digit relative to the chosen time origin, which is usually taken to coincide with the first bit. In general, the sequence i_0, i_1, i_2, i_3 .... has the representation $i_0 + i_1 D + i_2 D^2 + i_3 D^3 + ...$

A convolutional code over $GF(q)$ with a wordlength $k = (m+1)k_0$, a blocklength $n = (m+1)n_0$ and a constraint length $v = mk_0$ can be encoded by sets of finite impulse response (FIR) filters. Each set of filters consist of k_0 FIR filters in $GF(q)$. The input to the encoder is a sequence of k_0 symbols, and the output is a sequence of n_0 symbols. Figure 6.7 shows an encoder for a binary convolutional code with $k_0 = 1$ and $n_0 = 4$. Figure 6.8 shows a convolutional filter with $k_0 = 2$ and $n_0 = 4$.

Each of the FIR filters can be represented by a polynomial of degree $\leq m$. The input stream of symbols can also be represented by a polynomial. The operation of the filter can then simply be a multiplication of the two polynomials. Thus, the encoder (and hence the code) can be represented by a *set of polynomials* called the **generator polynomials** of the code. This set contains $k_0 n_0$ polynomials. The largest degree of a polynomial in this set of generator polynomials is m. We remind the reader that a block code was represented by a *single* generator polynomial. Thus we can define a **generator polynomial matrix** of size $k_0 \times n_0$ for a convolutional code as follows.

$$G(D) = [g_{ij}(D)] \tag{6.3}$$

Fig. 6.7 *A Convolutional Encoder in Terms of FIR Filters with $k_0 = 1$ and $n_0 = 4$.*

Fig. 6.8 *A Convolutional Encoder in Terms of FIR Filters with $k_0 = 2$ and $n_0 = 4$. The Rate of This Encoder is $R = ½$.*

Example 6.3 Consider the convolutional encoder given in Fig. 6.9.

Fig. 6.9 *The Rate ½ Convolutional Encoder with $G(D) = (D^2 + D + 1 \;\; D^2 + 1)$.*

The first bit of the output $a = i_{n-2} + i_{n-1} + i_n$ and the second bit of the output $b = i_{n-2} + i_n$, where i_{n-1} represents the input that arrived l time units earlier. Let the input stream of symbols be represented by a polynomial. We know that multiplying any polynomial by D corresponds to a single cyclic right-shift of the elements. Therefore,

$$g_{11}(D) = D^2 + D + 1 \text{ and } g_{12}(D) = D^2 + 1$$

and the generator polynomial matrix of this encoder can be written as

$$G(D) = [D^2 + D + 1 \;\; D^2 + 1].$$

Next, consider the encoder circuit shown in Fig. 6.10.

Fig. 6.10 *The Rate ½ Convolutional Encoder with $G(D) = (1 \;\; D^4 + 1)$.*

In this case, $a = i_n$ and $b = i_{n-4} + i_n$. Therefore, the generator polynomial matrix of this encoder can be written as

$$G(D) = [1 \;\; D^4 + 1].$$

Note that the first k_0 bits ($k_0 = 1$) of the codeword frame is identical to the information frame. Hence, this is a **Systematic Convolutional Encoder**.

Example 6.4 Consider the systematic convolutional encoder represented by the following circuit (Fig. 6.11).

Fig. 6.11 *The Rate 2/3 Convolutional Encoder for Example 6.4.*

The generator polynomial matrix of this encoder can be written as

$$G(D) = \begin{bmatrix} g_{11}(D) & g_{12}(D) & g_{13}(D) \\ g_{21}(D) & g_{22}(D) & g_{23}(D) \end{bmatrix} = \begin{bmatrix} 1 & 0 & D^3 + D + 1 \\ 0 & 1 & 0 \end{bmatrix}$$

It is easy to write the generator polynomial matrix by visual inspection. The element in the ith row and j^{th} column of the matrix represents the relation between the ith input bit and the jth output bit. To

write the generator polynomial for the (i^{th}, j^{th}) entry of the matrix, just trace the route from the ith input bit to the jth output bit. If no path exists, the generator polynomial is the zero polynomial, as in the case of $g_{12}(D)$, $g_{21}(D)$ and $g_{23}(D)$. If only a direct path exists without any delay elements, the value of the generator polynomial is unity, as in $g_{11}(D)$ and $g_{22}(D)$. If the route from the ith input bit to the jth output bit involves a series of memory elements (delay elements), represent each delay by an additional power of D, as in $g_{13}(D)$. Note that three of the generator polynomials in the set of generator polynomials are zero. When k_0 is greater than 1, it is not unusual for some of the generator polynomials to be the zero polynomials.

We can now give the formal definitions of the **Wordlength**, the **Blocklength** and the **Constraint Length** of a **Convolutional Encoder**.

Definition 6.6 Given the generator polynomial matrix $[g_{ij}(D)]$ of a convolutional code:

(i) The **Wordlength** of the code is

$$k = k_0 \max_{i,j}\left[\deg g_{ij}(D) + 1\right]. \tag{6.4}$$

(ii) The **Blocklength** of the code is

$$n = n_0 \max_{i,j}\left[\deg g_{ij}(D) + 1\right]. \tag{6.5}$$

(iii) The **Constraint Length** of the code is

$$v = \sum_{i=j}^{k_0} \max_{i,j}\left[\deg g_{ij}(D)\right]. \tag{6.6}$$

Recall that the input message stream $i_0, i_1, i_2, i_3 \ldots$ has the polynomial representation $I(D) = i_0 + i_1 D + i_2 D^2 + i_3 D^3 + \ldots + i_{k0} D^{k0}$ and the codeword polynomial can be written as $C(D) = c_0 + c_1 D + c_2 D^2 + c_3 D^3 + \ldots + c_{n0} D^{n0}$. The encoding operation can simply be described as vector matrix product,

$$C(D) = I(D)\boldsymbol{G}(D) \tag{6.7}$$

or equivalently,

$$c_j(D) = \sum_{i=1}^{k_0} i_l(D) g_{l,j}(D). \tag{6.8}$$

Observing that the encoding operation can simply be described as vector matrix product, it can be easily shown that convolutional codes belong to the class of linear codes (exercise).

Definition 6.7 A **Parity Check Matrix** $\boldsymbol{H}(D)$ is an $(n_0 - k_0)$ by n_0 matrix of polynomials that satisfies

$$\boldsymbol{G}(D)\boldsymbol{H}(D)^T = \boldsymbol{0} \tag{6.9}$$

and the **Syndrome Polynomial** vector which is a $(n_0 - k_0)$-component row vector is given by

$$\boldsymbol{s}(D) = \boldsymbol{v}(D)\boldsymbol{H}(D)^T \tag{6.10}$$

Definition 6.8 A **Systematic Encoder** for a convolutional code has the generator polynomial matrix of the form

$$\boldsymbol{G}(D) = [\boldsymbol{I} \mid \boldsymbol{P}(D)] \tag{6.11}$$

where $\boldsymbol{I}$ is a k_0 by k_0 identity matrix and $\boldsymbol{P}(D)$ is a k_0 by $(n_0 - k_0)$ matrix of polynomials. The parity check polynomial matrix for a systematic convolutional encoder is

$$\boldsymbol{H}(D) = [-\boldsymbol{P}(D)^T \mid \boldsymbol{I}] \tag{6.12}$$

where $\boldsymbol{I}$ is a $(n_0 - k_0)$ by $(n_0 - k_0)$ identity matrix. It follows that

$$\boldsymbol{G}(D)\boldsymbol{H}(D)^T = 0 \tag{6.13}$$

Definition 6.9 A convolutional code whose generator polynomials $g_1(D), g_2(D), \ldots, g_{n0}(D)$ satisfy

$$GCD[g_1(D), g_2(D), \ldots, g_{n0}(D)] = x^a \tag{6.14}$$

for some a is called a **Non-Catastrophic Convolutional Code**. Otherwise it is called a **Catastrophic Convolutional Code**.

Without loss of generality, one may take $a = 0$, i.e., $x^a = 1$. Thus the task of finding a non catastrophic convolutional code is equivalent to finding a good set of relatively prime generator polynomials. Relatively prime polynomials can be easily found by computer searches. However, what is difficult is to find a set of relatively prime generator polynomials that have good error correcting capabilities.

Example 6.5 All systematic codes are non-catastrophic because for them $g_1(D) = 1$ and therefore,

$$GCD[1, g_2(D)], \ldots, g_{n_0}(D)] = 1$$

Thus the systematic convolutional encoder represented by the generator polynomial matrix

$$\boldsymbol{G}(D) = [1 \; D^4 + 1]$$

is **non-catastrophic.**

Consider the following generator polynomial matrix of a binary convolutional encoder

$$\boldsymbol{G}(D) = [D^2 + 1 \; D^4 + 1]$$

We observe that $(D^2 + 1)^2 = D^4 + (D^2 + D^2) + 1 = D^4 + 1$ for binary encoder (modulo 2 arithmetic). Hence, $GCD[g_1(D), g_2(D)] = D^2 + 1 \neq 1$. Therefore, this is a **catastrophic encoder**.

Next, consider the following generator polynomial matrix of a non-systematic binary convolutional encoder

$$G(D) = [D^2 + 1 \quad D^4 + D^2 + 1]$$

The two generator polynomials are relatively prime, i.e., $GCD[g_1(D), g_2(D)] = 1$.

Hence this represents a **non-catastrophic** convolutional encoder.

In the next section, we see that the distance notion of convolutional codes is an important parameter that determines the number of errors a convolutional code can correct.

6.4 DISTANCE NOTIONS FOR CONVOLUTIONAL CODES

Recall that, for block codes, the concept of (Hamming) Distance between two codewords provides a way of quantitatively describing how different the two vectors are and how as a good code must possess the maximum possible minimum distance. Convolutional codes also have a distance concept that determines how good the code is.

When a codeword from a convolutional encoder passes through a channel, errors occur from time to time. The job of the decoder is to correct these errors by processing the received vector. In principle, the convolutional codeword is infinite in length. However, the decoding decisions are made on codeword *segments* of a finite length. The number of symbols that the decoder can store is called the **Decoding Window Width**. Regardless of the size of these finite segments (decoding window width), the previous frames affect the current frame because of the memory of the encoder. In general, one gets a better performance by increasing the decoding window width, but one eventually reaches a point of diminishing return.

Most of the decoding procedures for decoding convolutional codes work by focussing on the errors in the first frame. If this frame can be corrected and decoded, then the first frame of information is known at the receiver end. The effect of these information symbols on the subsequent information frames can be computed and subtracted from subsequent codeword frames. Thus the problem of decoding the second codeword frame is the same as the problem of decoding the first frame.

We extend this logic further. If the first f frames have been decoded successfully, the problem of decoding the $(f+1)^{th}$ frame is the same as the problem of decoding the first frame. But what happens if a frame in-between was not decoded correctly? If it is possible for a single decoding error event to induce an infinite number of *additional* errors, then the decoder is said to be subject to **Error Propagation**. In the case where the decoding algorithm is responsible for error propagation, it is termed as **Ordinary Error Propagation**. In the case where the poor choice of catastrophic generator polynomials cause error propagation, we call it **Catastrophic Error Propagation**.

Definition 6.10 The lth **minimum distance** d_l^* of a convolutional code is equal to the smallest Hamming Distance between any two initial codeword segments l frame long that are not identical in the initial frame. If $l = m + 1$, then this $(m+1)^{th}$ minimum distance is called the **Minimum Distance** of the code and is denoted by d^*, where m is the number of information frames that can be stored in the memory of the encoder. In literature, the minimum distance is also denoted by d_{min}.

We note here that a convolutional code is a linear code. Therefore, one of the two codewords used to determine the minimum distance of the code can be chosen to be the all zero codeword. The lth minimum distance is then equal to the weight of the smallest-weight codeword segement l frames long that is non zero in the first frame (i.e., different from the all zero frame).

Suppose the lth minimum distance of a convolutional code is d_l^*. The code can correct t errors occurring in the first l frames provided

$$d_l^* \geq 2t + 1 \tag{6.15}$$

Next, put $l = m + 1$, in which case $d_l^* = d_{m+1}^* = d^*$. The code can correct t errors occurring in the first blocklength $n = (m+1)n_0$ provided

$$d^* \geq 2t + 1 \tag{6.16}$$

Example 6.6 Consider the convolutional encoder of Example 6.1 (Fig. 6.3). The Trellis Diagram for the encoder is given in Fig. 6.12.

Fig. 6.12 *The Trellis Diagram for the Convolutional Encoder of Example 6.1.*

In this case $d_1^* = 2$, $d_2^* = 3$, $d_3^* = 5$, $d_4^* = 5$, ... We observe that $d_i^* = 5$ for $i \geq 3$. For this encoder, $m = 2$. Therefore, the minimum distance of the code is $d_3 = d^* = 5$. This code can correct $(d^* - 1)/2 = 2$ random errors that occur in one blocklength, $n = (m+1)n_0 = 6$.

Definition 6.11 The **Free Distance** of a convolutional code is given by

$$d_{free} = \max_{l} [d_j] \tag{6.17}$$

It follows that $d_{m+1} \le d_{m+2} \le \ldots \le d_{free}$.

The term d_{free} was first coined by Massey in 1969 to denote a type of distance that was found to be an important parameter for the decoding techniques of convolutional codes. Since, d_{free} represents the minimum distance between arbitrarily long (possibly infinite) encoded sequences, d_{free} is also denoted by d_∞ in literature. The parameter d_{free} can be directlycalculated from the trellis diagram. The free distance d_{free} is the minimum weight of a path that deviates from the all zero path and later merges back into the all zero path at some point further down the trellis as depicted in Fig. 6.13. Searching for a code with large minimumdistance and large free distance is a tedious process, and is often done using a computer.Clever techniques have been designed that reduce the effort by avoiding exhaustive searches. Most of the good convolutional codes known today have been discovered by computer searches.

Definition 6.12 The **free length** n_{free} of a convolutional code is the length of the non-zero segment of a smallest weight convolutional codeword of non-zero weight. Thus, $d_l = d_{free}$ if $l = n_{free}$, and $d_l < d_{free}$ if $l < n_{free}$. In literature, n_{free} is also denoted by n_∞.

Fig. 6.13 *The Free Distance d_{free} path.*

Example 6.7 Consider the convolutional encoder of Example 6.1 (Fig. 6.3). For this encoder, $d_{free} = 5$. There are usually more than one pair of paths that can be used to calculate d_{free}. The two paths that have been used to calculate d_{free} are shown in Fig. 6.14 by double lines. In this example, $d_{min} = d_{free}$.

Fig. 6.14 *Calculating d_{free} in the Trellis Diagram.*

The free length of the convolutional code is $n_{free} = 6$. In this example, the n_{free} is equal to the blocklength n of the code. In general it can be longer than the blocklength.

6.5 THE GENERATING FUNCTION

The performance of a convolutional code depends on its free distance, d_{free}. Since convolutional codes are a sub-class of linear codes, the set of Hamming distances between coded sequences is the same as the set of distances of the coded sequences from the all-zero sequence. Therefore, we can consider the distance structure of the convolutional codes with respect to the all-zero sequence without loss of generality. In this section, we shall study an elegant method of determining the free distance, d_{free}, of a convolutional code.

To find d_{free} we need the set of paths that diverge from the all-zero path and merge back at a later time. The brute force (and time consuming, not to mention, exasperating) method is to determine the distances of all possible paths from the trellis diagram. Another way to find out the d_{free} of a convolutional code is use the concept of a **generating function**, whose expansion provides all the distance information directly. The generating function can be understood by the following example.

Example 6.8 Consider again the convolutional encoder of Example 6.1 (Fig. 6.3). The state diagram of the encoder is given in Fig. 6.4. We now construct a modified state diagram as shown in Fig. 6.15.

The branches of this modified state diagram are labelled by branch gain D^i, $i = 0, 1, 2$, where the exponent of D denotes the Hamming Weight of the branch. Note that the self loop ar S_0 has been neglected as it does not contribute to the distance property of the code. Circulating around this loop simply generates the all zero sequence. Note that S_0 has been split into two states, initial and final. Any path that diverges from state S_0 and later merges back to S_0 can be thought of equivalently as traversing along the branches of this modified state diagram, starting from the

initial S_0 and ending at the final S_0. Hence this modified state diagram encompasses all possible paths that diverge from and then later merge back to the all zero path in the trellis diagram.

Fig. 6.15 *The Modified State Diagram of the Convolutional Encoder Shown in Fig. 6.3.*

We can find the distance profile of the convolutional code using the state equations of this modified state diagram. These state equations are

$$X_1 = D_2 + X_2,$$
$$X_2 = DX_1 + DX_3,$$
$$X_3 = DX_1 + DX_3,$$
$$T(D) = D_2X_2, \tag{6.18}$$

where X_i are dummy variables. Upon solving these equations simultaneously, we obtain the generating function

$$T(D) = \frac{D^5}{1-2D}$$
$$= D^5 + 2D^6 + 4D^7 + \cdots + 2^k D^{k+5} + \cdots$$
$$= \sum_{d=5}^{\infty} a_d D^d \tag{6.19}$$

Note that the expression for $T(D)$ can alxso be (easily) obtained by the **Mason's Gain Formula**, which is well known to the students of **Digital Signal Processing**.

Following conclusions can be drawn from the series expansion of the generating function:

(i) There are an infinite number of possible paths that diverge from the all zero path and later merge back again (this is also intuitive).

(ii) There is only one path with Hamming Distance 5, two paths with Hamming Distance 6, and in general 2^k paths with Hamming Distance $k + 5$ from the all zero path.

(iii) The free Hamming Distance d_{free} for this code is 5. There is only one path corresponding to d_{free}. Example 6.7 explicitly illustrates the pair of paths that result in $d_{free} = 5$.

We now introduce two new labels in the modified state diagram. To enumerate the length of a given path, the label L is attached to each branch. So every time we traverse along a branch we increment the path length counter. We also add the label I^i, to each branch to enumerate the Hamming Weight of the input bits associated with each branch of the modified state diagram (see Fig. 6.16).

Fig. 6.16 *The Modified State Diagram to Determine the Augmented Generating Function.*

The state equations in this case would be

$$X_1 = D_2LI + LIX_2,$$
$$X_2 = DLX_1 + DLX_3,$$
$$X_3 = DLIX_1 + DLIX_3,$$

And the **Augmented Generating Function** is

$$T(D) = D^2LX_2\,. \tag{6.20}$$

On solving these simultaneous equations, we obtain

$$T(D, L, I) = \frac{D^5L^3I}{1 - DL(L+1)I}$$
$$= D^5L^3I + D^6L^4(L+1)I^2 + \cdots + D^{k+5}L^{3+k}(L+1)^kI^{k+1}\cdots \tag{6.21}$$

Further conclusions from the series expansion of the augmented generating function are:

(i) The path with the minimum Hamming Distance of 5 has length equal to 3.

(ii) The input sequence corresponding to this path has weight equal to 1.

(iii) There are two paths with Hamming Distance equal to 6 from the all zero path. Of these, one has a path length of 4 and the other 5 (observe the power of L in the second term in the summation). Both these paths have an input sequence weight of 2.

In the next section, we study the matrix description of convolutional codes which is a bit more complicated than the matrix description of linear block codes.

6.6 MATRIX DESCRIPTION OF CONVOLUTIONAL CODES

A convolutional code can be described as consisting of an infinite number of infinitely long codewords and which (visualize the trellis diagram) belongs to the class of linear codes. They can be described by an infinite generator matrix. As can be expected, the matrix description of convolutional codes is messier than that of the linear block codes.

Let the generator polynomials of a Convolutional Code be represented by

$$g_{ij}(D) = \sum_l g_{ij}\, D^l \tag{6.22}$$

In order to obtain a generator matrix, the g_{ijl} coefficients are arranged in a matrix format. For each l, let G_l be a k_0 by n_0 matrix.

$$G_l = [g_{ijl}] \tag{6.23}$$

Then, the generator matrix for the Convolutional Code that has been truncated to a block code of blocklength n is

$$G^{(n)} = G^{(n)} = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m \\ 0 & G_0 & G_0 & \cdots & G_{m-1} \\ 0 & 0 & G_0 & \cdots & G_{m-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & G_0 \end{bmatrix} \tag{6.24}$$

where 0 is a k_0 by n_0 matrix of zeros and m is the length of the shift register used to generate the code. The generator matrix for the Convolutional Code is given by

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \cdots \\ & \vdots & \vdots & & & & & & & \vdots \end{bmatrix}. \tag{6.25}$$

The matrix extends infinitely far down and to the right. For a systematic convolutional code, the generator matrix can be written as

$$G = \left[\begin{array}{cc|cc|cc|c|cc|cc|cc|c} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} & 0 & P_m & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & I & P_0 & \cdots & 0 & P_{m-2} & 0 & P_{m-1} & 0 & P_m & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & P_{m-2} & 0 & P_{m-1} & \cdots \\ & & & & & & & & & \vdots & \vdots & 0 & P_{m-2} & \cdots \\ & & & & & & & & & \vdots & \vdots & \vdots & \vdots & \end{array}\right] \tag{6.26}$$

where I is a k_0 by k_0 identity matrix, 0 is a k_0 by k_0 matrix of zeros and $P_0, P_2, \ldots, P_m$ are k_0 by $(n_0 - k_0)$ matrices. The parity check matrix can then be written as

$$H = \begin{bmatrix} P_0^T & -I & & & & & & & & \cdots \\ P_1^T & 0 & P_0^T & -I & & & & & & \\ P_2^T & 0 & P_1^T & 0 & P_0^T & -I & & & & \\ \vdots & & \vdots & & \vdots & & & & & \vdots \\ P_m^T & 0 & P_{m-1}^T & 0 & P_{m-2}^T & 0 & \cdots & P_0^T & -I & \cdots \\ & & P_m^T & 0 & P_{m-1}^T & 0 & \cdots & & & \\ & & & & P_m^T & 0 & \cdots & & \cdots & \end{bmatrix} \tag{6.27}$$

Example 6.9 Consider the convolutional encoder shown in Fig. 6.17. Let us first write the generator polynomial matrix for this encoder. To do so, we just follow individual inputs to the outputs, one by one, and count the delays. The generator polynomial matrix is obtained as

$$G(D) = \begin{bmatrix} D + D^2 & D^2 & D + D^2 \\ D^2 & D & D \end{bmatrix}$$

Fig. 6.17 *A Rate 2/3 Convolutional Encoder.*

The generator polynomials are $g_{11}(D) = D + D^2$, $g_{12}(D) = D^2$, $g_{13}(D) = D + D^2$, $g_{21}(D) = D^2$, $g_{22}(D) = D$ and $g_{23}(D) = D$.

To write out the matrix G_0, we look at the constants (coefficients of D^0) in the generator polynomials. Since there are no constant terms in any of the generator polynomials,

$$G_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Next, to write out the matrix G_1, we look at the coefficients of D^1 in the generator polynomials. The 1st row, 1st column entry of the matrix G_1 corresponds to the coefficients of D^1 in $g_{11}(D)$. The 1st row, 2nd column entry corresponds to the coefficients of D^1 in $g_{12}(D)$, and so on. Thus,

$$G_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Similarly, we can write

$$G_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The generator matrix can now be written as

$$G = \left[\begin{array}{ccc|ccc|ccc|ccc|c} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & & 0 & 0 & \cdots \\ \hline & & & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & \\ & & & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \\ \hline & \vdots & & & & & & & & 1 & 0 & 1 & \\ & \vdots & & & \vdots & & & \vdots & & 0 & 1 & 1 & \cdots \end{array}\right]$$

Our next task is to look at an efficient decoding strategy for the convolutional codes. One of the very popular decoding methods, the **Viterbi Decoding Technique**, is discussed in detail.

6.7 VITERBI DECODING OF CONVOLUTIONAL CODES

There are three important decoding techniques for convolutional codes: Sequential Decoding, Threshold Decoding and the Viterbi Decoding. The **Sequential Decoding** technique was proposed by Wozencraft in 1957. Sequential Decoding has the advantage that it can perform very well with long-constraint-length convolutional codes, but it has a variable decoding time. **Threshold Decoding**, also known as **Majority Logic Decoding**, was proposed by Massey in 1963 in his doctoral thesis at MIT. Threshold decoders were the first commercially produced decoders for convolutional codes. **Viterbi Decoding** was developed by Andrew J. Viterbi in 1967 and in the late 1970s became the dominant technique for convolutional codes. Viterbi Decoding had the advantages of

(i) a highly satisfactory bit error performance,
(ii) high speed of operation,
(iii) ease of implementation,
(iv) low cost.

Threshold decoding lost its popularity specially because of its inferior bit error performance. It is conceptually and practically closest to block decoding and it requires the calculation of a set of syndromes, just as in the case of block codes. In this case, the syndrome is a sequence because the information and check digits occur as sequences.

Viterbi Decoding has the advantage that it has a fixed decoding time. It is well suited to hardware decoder implementation. But its computational requirements grow exponentially as a function of the constraint length, so it is usually limited in practice to constraint lengths of

$v = 9$ or less. As of early 2000, some leading companies claimed to have produced a $v = 9$ Viterbi decoder that operates at rates up to 2 Mbps.

Since the time when Viterbi proposed his algorithm, other researchers have expanded on his work by finding good convolutional codes, exploring the performance limits of the technique, and varying decoder design parameters to optimize the implementation of the technique in hardware and software. The Viterbi Decoding algorithm is also used in decoding Trellis Coded Modulation (TCM), the technique used in telephone-line modems to squeeze high ratios of bits per-second to Hertz out of 3 kHz-bandwidth analog telephone lines. We shall see more of TCM in the next chapter. For years, convolutional coding with Viterbi Decoding has been the predominant FEC (**Forward Error Correction**) technique used in space communications, particularly in geostationary satellite communication networks such as VSAT (very small aperture terminal) networks. The most common variant used in VSAT networks is rate 1/2 convolutional coding using a code with a constraint length $v = 7$. With this code, one can transmit binary or quaternary phase-shift-keyed (BPSK or QPSK) signals with at least 5 dB less power than without coding. That is a reduction in Watts of more than a factor of three! This is very useful in reducing transmitter and antenna cost or permitting increased data rates given the same transmitter power and antenna sizes.

We will now consider how to decode convolutional codes using the Viterbi Decoding algorithm. The nomenclature used here is that we have a message vector $\boldsymbol{i}$ from which the encoder generates $\boldsymbol{a}$ code vector $\boldsymbol{c}$ that is sent across a discrete memoryless channel. The received vector $\boldsymbol{r}$ may differ from the transmitted vector $\boldsymbol{c}$ (unless the channel is ideal or we are very lucky!). The decoder is required to make an estimate of the message vector. Since there is a one to one correspondence between code vector and message vector, the decoder makes an estimate of the code vector.

Optimum decoding will result in a minimum probability of Decoding error. Let $p(\boldsymbol{r}|\boldsymbol{c})$ be the conditional probability of receiving $\boldsymbol{r}$ given that $\boldsymbol{c}$ was sent. We can state that the optimum decoder is the maximum likelihood decoder with a decision rule to choose the code vector estimate $\hat{\boldsymbol{c}}$ for which the log-likelihood function $\ln p(\boldsymbol{r}|\boldsymbol{c})$ is maximum.

If we consider a BSC where the vector elements of $\boldsymbol{c}$ and $\boldsymbol{r}$ are denoted by c_i and r_i, then, we have

$$p(\boldsymbol{r}|\boldsymbol{c}) = \sum_{i=1}^{N} p(r_i|c_i), \tag{6.28}$$

and hence, the log-likelihood function equals

$$\ln p(\boldsymbol{r} \mid \boldsymbol{c}) = \sum_{i=1}^{N} \ln p(r_i \mid c_i) \tag{6.29}$$

Let us assume

$$p(r_i|c_i) = \begin{cases} p, & r_i \neq c_i \\ 1-p & r_i = c_i \end{cases} \tag{6.30}$$

If we suppose that the received vector differs from the transmitted vector in exactly d positions (the Hamming Distance between vectors $\boldsymbol{c}$ and $\boldsymbol{r}$), we may rewrite the log-likelihood function as

$$\ln p(\boldsymbol{r} \mid \boldsymbol{c}) = d \ln p + (N - d) \ln(1 - p)$$
$$= d \ln \left(\frac{p}{1-p} \right) + N \ln (1 - p) \tag{6.31}$$

We can assume the probability of error $p < ½$ and we note that $N \ln(1 - p)$ is a constant for all code vectors. Now we can make the statement that *the maximum likelihood decoding rule for a Binary Symmetric Channel is to choose the code vector estimate $\hat{\boldsymbol{c}}$ that minimizes the Hamming Distance between the received vector $\boldsymbol{r}$ and the transmitted vector $\boldsymbol{c}$.*

For Soft Decision Decoding in Additive White Gaussian Noise (AWGN) channel with single sided noise power N_0, the likelihood function is given by

$$p(\boldsymbol{r} \mid \boldsymbol{c}) = \prod_{i=1}^{N} \frac{1}{\sqrt{\pi N_0}} e^{\frac{|r_i - c_i|^2}{N_0}}$$
$$= \left(\frac{1}{\pi N_0} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{N_0} \sum_{i=1}^{N} |r_i - c_i|^2 \right) \tag{6.32}$$

Thus the maximum likelihood decoding rule for the AWGN channel with Soft Decision Decoding is to *minimize the squared Euclidean Distance* between $\boldsymbol{r}$ and $\boldsymbol{c}$. This squared Euclidean Distance is given by

$$d_E^2(\boldsymbol{r} \mid \boldsymbol{c}) = \sum_{i=1}^{N} |r_i - c_i|^2 \tag{6.33}$$

Viterbi Decoding works by choosing that trial information sequence, the encoded version of which is *closest* to the received sequence. Here, Hamming Distance will be used as a measure of proximity between two sequences. The Viterbi Decoding procedure can be easily understood by the following example.

Example 6.10 Consider the rate 1/3 convolutional encoder given in Fig. 6.18 and the corresponding trellis diagram.

(Contd...)

Fig. 6.18 *A Rate 1/3 Convolutional Encoder and Its Trellis Diagram.*

Suppose the transmitted sequence was an all zero sequence. Let the received sequence be

r= 010000100001 ...

Since it is a 1/3 rate encoder, we first segment the received sequence in groups of three bits (because $n_0 = 3$), i.e.,

r = 010 000 100 001 ...

The task at hand is to find out the most likely path through the trellis. Since a path must pass through nodes in the trellis, we will try to find out which nodes in the trellis belong to the most likely path. At any time, every node has two incoming branches (→). We simply determine which of these two branches belongs to a more likely path (and discard the other). We make this decision based on some metric (Hamming Distance). In this way we retain just one path per node and the metric of that path. In this example, we retain only four paths as we progress with our decoding (since we have only 4 states in our trellis).

Let us consider the first branch of the trellis which is labelled 000. We find the Hamming distance between this branch and the first received framelength, 010. The Hamming distance d (000, 010) = 1. Thus the metric for this first branch is 1, and is called the **Branch Metric**. Upon reaching the top node from the starting node, this branch has accumulated a metric = 1. Next, we compare the received framelength with the lower branch, which terminates at the second node from the top. The Hamming Distance in this case is d (111, 010) = 2. Thus, the metric for this first branch is 2. At each node we write the total metric accumulated by the path, called the **Path Metric**. The path metrics are marked by circled numbers in the trellis diagram in Fig. 6.19. At the subsequent stages of decoding when two paths terminate at every node, we will retain the path with the smaller value of the metric.

Fig. 6.19 *The Path Metric after the 1st Step of Viterbi Decoding.*

We, now, move to the next stage of the trellis. The Hamming Distance between the branches are computed with respect to the second frame received, 000. The branch metrics for the two branches emanating from the topmost node are 0 and 3. The branch metrics for the two branches emanating from the second node are 2 and 1. The total path metric is marked by circled numbers in the trellis diagram shown in Fig. 6.20.

Fig. 6.20 *The Path Metric after the 2nd Step of Viterbi Decoding.*

We now proceed to the next stage. We again compute the branch metrics and add them to the respective path metrics to get the new path metrics. Consider the topmost node at this stage. Two branches terminate at this node. The path coming from node 1 of the previous stage has a path metric of 2 and the path coming from node 1 of the previous stage has a path metric of 6. The path with a lower metric is retained and the other discarded. The trellis diagram shown in Fig. 6.21 gives the *surviving paths* (double lines) and the path metrics (circled numbers). Viterbi called these surviving paths as **Survivors.** It is interesting to note that node 4 receives two paths with equal path metrics. We have arbitrarily chosen one of them as the surviving path (by tossing a fair coin!).

Fig. 6.21 *The Path Metric after the 3rd Step of Viterbi Decoding.*

We continue this procedure for Viterbi decoding for the next stage. The final branch metrics and path metrics are shown in Fig. 6.22. At the end we pick the path with the minimum metric. This path corresponds to the all zero path. Thus the decoding procedure has been able to correctly decode the received vector.

Fig. 6.22 *The Path Metric after the 4th Step of Viterbi Decoding.*

The minimum distance for this code is $d^* = 6$. The number of errors that it can correct per frame length is equal to

$$t = \lfloor (d^* - 1)/2) \rfloor = \lfloor (6 - 1)/2 \rfloor = 2.$$

In this example, the maximum number of errors per framelength was 1.

Consider the set of surviving paths at the rth frame time. If all the surviving paths cross through the same nodes then a decision regarding the most likely path transmitted can be made up to the point where the nodes are common. To build a practical Viterbi Decoder, one must choose a decoding window width w, which is usually several times as big as the blocklength. At a given frame time, f, the decoder examines all the surviving paths to see if they agree in the first

branch. This branch defines a decoded information frame and is passed out of the decoder. In the previous example of Viterbi Decoding, we see that by the time the decoder reaches the 4th frame, all the surviving paths agree in their first decoded branch (called a well-defined decision). The decoder drops the first branch (after delivering the decoded frame) and takes in a new frame of the received word for the next iteration. If again, all the surviving paths pass through the same node of the oldest surviving frame, then this information frame is decoded. The process continues in this way indefinitely.

If a long enough decoding window w is chosen, then a well-defined decision can be reached almost always. A well designed code will lead to correct decoding with a high probability. Note that a well designed code carries meaning only in the context of a particular channel. The random errors induced by the channel should be within the error correcting capability of the code. The Viterbi decoder can be visualized as a sliding window through which the trellis is viewed (see Fig. 6.23). The window slides to the right as new frames are processed. The surviving paths are marked on the portion of the trellis which is visible through the window. As the window slides, new nodes appear on the right, and some of the surviving paths are extended to these new nodes while the other paths disappear.

Fig. 6.23 *The Viterbi Decoder as a Sliding Window through which the Trellis is Viewed.*

If the surviving paths do not go through the same node, we label it a **Decoding Failure**. The decoder can break the deadlock using any arbitrary rule. To this limited extent, the decoder becomes an incomplete decoder. Let us revert back to the previous example. At the 4th stage, the surviving paths could as well be chosen as shown in Fig. 6.24, which will render the decoder as an incomplete decoder.

Fig. 6.24 *Example of an Incomplete Decoder in Viterbi Decoding Process.*

It is possible that in some cases the decoder reaches a well-defined decision, but a wrong one! If this happens, the decoder has no way of knowing that it has taken a wrong decision. Based on this wrong decision, the decoder will take more wrong decisions. However, if the code is non-catastrophic, the decoder will recover from the errors.

The next section deals with some **Distance Bounds** for convolutional codes. These bounds will help us compare different convolutional coding schemes.

6.8 DISTANCE BOUNDS FOR CONVOLUTIONAL CODES

Upper bounds can be computed on the minimum distance of a convolutional code that has a rate $R = \frac{k_0}{n_0}$ and a constraint length $v = mk_0$. These bounds are similar in nature and derivation to those for block codes, with block length corresponding to constraint length. However, as we shall see, the bounds are not very tight. These bounds just give us a rough idea of how good the code is. Here we present the bounds (without proof) for binary codes.

For rate R and constraint length v, let d be the largest integer that satisfies

$$H\left(\frac{d}{n_0 v}\right) \le 1 - R \tag{6.34}$$

Then at least one binary convolutional code exists with minimum distance d for which the above inequality holds. Here $H(x)$ is the familiar entropy function for a binary alphabet

$$H(x) = -x\log_2 x - (1-x)\log_2(1-x),\ 0 \le x \le 1$$

For a binary code with $R = 1/n_0$ the minimum distance d_{min} satisfies

$$d_{min} \le \lfloor (n_0 v + n_0)/2 \rfloor \tag{6.35}$$

where $\lfloor I \rfloor$ denotes the largest integer less than or equal to I.

An upper bound on d_{free} is given by (Heller, 1968)

$$d_{free} = \min_{j \ge 1} \left\lfloor \frac{n_0}{2}\frac{2^j}{2^j - 1} v + j - 1 \right\rfloor \tag{6.36}$$

To calculate the upper bound, the right hand side should be plotted for different integer values of j. The upper bound is the minimum of this plot.

Example 6.10 Let us apply the distance bounds on the convolutional encoder given in Example 6.1. We will first apply the bound given by (6.34). For this encoder, $k_0 = 1$, $n_0 = 2$, $R = 1/2$ and $v = 2$.

$$H\left(\frac{d}{n_0 v}\right) = H(d/4) \le 1 - R = 1/2 \Rightarrow H(d/4) \le 0.5$$

But we have,

$$H(0.11) = -0.11 \log_2 0.11 - (1 - 0.11) \log_2 (1 - 0.11) = 0.4999, \text{ and}$$
$$H(0.12) = -0.12 \log_2 0.12 - (1 - 0.12) \log_2 (1 - 0.12) = 0.5294.$$

Therefore, $d/4 \leq .0.11$, or $d \leq 0.44$

The largest integer d that satisfies this bound is $d = 0$. This implies that at least one binary convolutional code exists with minimum distance $d = 0$. This statement does not say much (i.e., the bound is not strict enough for this encoder)!

Next, consider the encoder shown in Fig. 6.25.

Fig. 6.25 *Convolutional Encoder for Example 6.10.*

For this encoder,

$$G(D) = [1 \; D + D^2 \; D + D^2 + D^3],$$
$$k_0 = 1, \; n_0 = 3, \; R = 1/3 \text{ and } v = 3.$$

$$H\left(\frac{d}{n_0 v}\right) = H(d/9) \leq 1 - R = 2/3 \Rightarrow H(d/9) \leq 0.6666$$

But we have,

$$H(0.17) = -0.17 \log_2 0.17 - (1 - 0.17) \log_2 (1 - 0.17) = 0.6577, \text{ and}$$
$$H(0.18) = -0.18 \log_2 0.18 - (1 - 0.18) \log_2 (1 - 0.18) = 0.6801$$

Therefore, $d/9 \leq 0.17$, or $d \leq 1.53$.

The largest integer d that satisfies this bound is $d = 1$. Then at least one binary convolutional code exists with minimum distance $d = 1$. This is a very loose bound.

Let us now evaluate the second bound, given by (6.35).

$$d_{min} \leq \lfloor (n_0 v + n_0)/2 \rfloor = \lfloor (9+3)/2 \rfloor = 6$$

This gives us $d_{min} = 6$, which is a good upper bound as seen from the trellis diagram for the encoder (Fig. 6.26). Since $n_0 = 3$, every branch in the trellis is labelled by 3 bits. The two paths that have been used to calculate d_{min} are shown in Fig. 6.26 by double lines. In this example, $d_{min} = d_{free} = 6$.

Fig. 6.26 *The Trellis Diagram for the Convolutional Encoder Given in Fig. 6.25.*

Next, we determine the Heller Bound on d_{free}, as given by (6.36). The plot of the function

$$d(j) = \left\lfloor (n_0/2)\left(2^j/(2^j - 1)\right)(v + j - 1) \right\rfloor$$

for different integer values of j is given in Fig. 6.27.

Fig. 6.27 *The Heller Bound Plot.*

From Fig. 6.27, we see that the upper bound on the free distance of the code is $d_{free} \leq 8$. This is a good upper bound. The actual value of $d_{free} = 6$.

6.9 PERFORMANCE BOUNDS

One of the useful performance criteria for convolutional codes is the bit error probability P_b. The bit error probability or the bit error rate (a misnomer!) is defined as the expected number of decoded information bit errors per information bit. Instead of obtaining an exact expression for P_b, typically, an upper bound on the error probability is calculated. We will first determine the **First Event Error Probability**, which is the probability of error for sequences that merge with the all zero (correct) path for the first time at a given node in the trellis diagram.

Since convolutional codes are linear, let us assume that the all zero codeword is transmitted. An error will be made by the decoder if it chooses the incorrect path c' instead of the all zero path. Let c' differ from the all zero path in d bits. Therefore, a wrong decision will be made by the maximum likely decoder if more than $\left\lfloor \frac{d}{2} \right\rfloor$ errors occur, where $\lfloor x \rfloor$ is the largest integer less than or equal to x. If the channel transition probability is p, then the probability of error can be upper bounded as follows.

$$P_d \le \left[2\sqrt{p(1-p)}\right]^d \tag{6.37}$$

Now, there would be many paths with different distances that merge with the correct path at a given time for the first time. The upper bound on the first event error probability can be obtained by summing the error probabilities of all such possible paths:

$$P_e \le \sum_{d=d_{free}}^{\infty} a_d P_d \tag{6.38}$$

where, a_d is the number of codewords of Hamming Distance d from the all zero codeword. Comparing (6.19) and (6.38) we obtain

$$P_e \le T(D)\big|_{D=2\sqrt{p(1-p)}} \tag{6.39}$$

The bit error probability, P_b, can now be determined as follows. P_b can be upper bounded by weighting each pairwise error probability, P_d in (6.37) by the number of incorrectly decoded information bits for the corresponding incorrect path n_d. For a rate k/n encoder, the average P_b

$$P_b \le \frac{1}{k} \sum_{d=d_{free}}^{\infty} a_d n_d P_d \tag{6.40}$$

It can be shown that

$$\left.\frac{\partial T(D,I)}{\partial I}\right|_{I=1} = \sum_{d=d_{free}}^{\infty} a_d n_d D^d \tag{6.41}$$

Thus,

$$P_b \le \frac{1}{k} \left.\frac{1\partial T(D,I)}{\partial I}\right|_{I=1,\, D=2\sqrt{(1-p)}} \tag{6.42}$$

6.10 KNOWN GOOD CONVOLUTIONAL CODES

In this section, we shall look at some known good convolutional codes. So far, only a few constructive classes of convolutional codes have been reported. There exists no class with an algebraic structure comparable to the t-error correcting BCH codes. No constructive methods

exist for finding convolutional codes of long constraint length. Most of the codes presented here have been found by computer searches.

Initial work on short convolutional codes with maximal free distance was reported by Odenwalder (1970) and Larsen (1973). A few of the codes are listed in Tables 6.2, 6.3 and 6.4 for code rates 1/2, 1/3 and 1/4 respectively. The generator is given as the sequence 1, $\gamma_1^{(i)}, \gamma_2^{(i)} \cdots$ where

$$g_i(D) = 1 + \gamma_1^{(i)} D + \gamma_2^{(i)} D^2 + \cdots + \gamma_{K-1}^{(i)} D^{K-1}. \tag{6.43}$$

For example, the octal notation for the generators of the $R = ½$, $v = 4$ encoders are 15 and 17 (see Table 6.2). The octal 15 can be deciphered as 15 = 1-5 = 1-101. Therefore,

$$g_1(D) = 1 + (1)D + (0)D^2 + (1)D^3 = 1 + D + D^3.$$

Similarly,

$$17 = 1\text{-}7 = 1\text{-}111.$$

Therefore,

$$g_2(D) = 1 + (1)D + (1)D^2 + (1)D^3 = 1 + D + D^2 + D^3, \text{ and}$$

$$G(D) = [1 + D + D^3 \;\; 1 + D + D^2 + D^3].$$

Table 6.2 *Rate ½ codes with maximum free distance*

	v	n	Generators (octal)	d_{free}	Heller Bound
			Non Catastrophic		
3	6	5	7	5	5
4	8	15	17	6	6
5	10	23	35	7	8
6	12	53	75	8	8
7	14	133	171	10	10
			Catastrophic		
5	10	27	35	8	8
12	24	5237	6731	16	16
14	28	21645	37133	17	17

Table 6.3 *Rate 1/3 codes with maximum free distance.*

v	n	Generators (octal)			d_{free}	Heller Bound
3	9	5	7	7	8	8
4	12	13	17	17	10	10
5	15	25	37	37	12	12
6	18	47	75	75	13	13
7	21	133	175	175	15	15

Table 6.4 *Rate 1/4 codes with maximum free distance*

v	n	Generators (Octal)				d_{free}	Heller Bound
3	12	5	7	7	7	10	10
4	16	13	15	15	17	15	15
5	20	25	27	33	37	16	16
6	24	53	67	71	75	18	18
7	28	135	135	147	163	20	20

Next, we study an interesting class of codes, called Turbo Codes, which lie somewhere between linear block codes and convolutional codes.

6.11 TURBO CODES

Turbo Codes were introduced in 1993 at the *International Conference on Communications* (ICC) by Berrou, Glavieux and Thitimajshima in their paper "Near Shannon Limit Error Correction Coding and Decoding–Turbo-Codes". In this paper, they quoted a BER performance of 10^{-5} at an E_b/N_0 of 0.7 dB using only a 1/2 rate code, generating tremendous interest in the field. Turbo Codes perform well in the low SNR scenario. At high SNRs, some of the traditional codes like the Reed–Solomon Code have comparable or better performance than Turbo Codes.

Even though Turbo Codes are considered as Block Codes, they do not exactly work like block codes. Turbo Codes are actually a quasi mix between Block and Convolutional Codes. They require, like a block code, that the whole block be present before encoding can begin. However, rather than computing parity bits from a system of equations, they use shift registers just like Convolutional Codes.

Turbo Codes typically use at least two convolutional component encoders and two ***maximum aposteriori*** (MAP) algorithm component decoders in the Turbo codes. This is known as **concatenation**. Three different arrangements of turbo codes are **Parallel Concatenated Convolutional Codes** (PCCC), **Serial Concatenated Convolutional Codes** (SCCC), and **Hybrid Concatenated Convolutional Codes** (HCCC). Typically, Turbo Codes are arranged like the PCCC. An example of a PCCC Turbo encoder given in Fig. 6.28 shows that two encoders run in parallel.

Fig. 6.28 *Block Diagram of a Rate 1/3, PCCC Turbo Encoder.*

One reason for the better performance of Turbo codes is that they produce high weight code words. For example, if the input sequence (U_k) is originally low weight, the systematic (X_k) and parity 1 (Y_k^1) outputs may produce a low weight codeword. However, the parity 2 output (Y_k^2) is less likely to be a low weight codeword due to the interleaver in front of it. The interleaver shuffles the input sequence, U_k, in such a way that when introduced to the second encoder, it is more likely to produce a high weight codeword. This is ideal for the code because high weight codewords result in better decoder performance. Intuitively, when one of the encoders produces a 'weak' codeword, the other encoder has a low probability of producing another 'weak' codeword because of the interleaver. The concatenated version of the two codewords is, therefore, a 'strong' codeword. Here, the expression 'weak' is used as a measure of the average Hamming Distance of a codeword from all other codewords.

Although the encoder determines the *capability* for error correction, it is the decoder that determines the *actual performance*. The performance, however, depends upon which algorithm is used. Since Turbo Decoding is an iterative process, it requires a soft output algorithm like the *maximum a-posteriori* algorithm (MAP) or the **Soft Output Viterbi Algorithm** (SOVA) for decoding. Soft output algorithms out-perform hard decision algorithms because they have available a better estimate of what the sent data actually was. This is because soft output yields a gradient of information about the computed information bit rather than just choosing a 1 or 0 like hard output. A typical Turbo Decoder is shown in Fig. 6.29.

The MAP algorithm is often used to estimate the most likely information bit to have been transmitted in a coded sequence. The MAP algorithm is favoured because it outperforms other algorithms, such as the SOVA, under low SNR conditions. The major drawback, however, is that it is more complex than most algorithms because of its focus on each individual bit of information. Research in the area (in late 1990s) has resulted in great simplification of the MAP algorithm.

Fig. 6.29 *Block Diagram of a Turbo Decoder.*

A Turbo Decoder generally uses the MAP algorithm in at least one of its component decoders. The decoding process begins by receiving partial information from the channel (X_k and Y_k^1) and passing it to the first decoder. The rest of the information, parity 2 (Y_k^2), goes to the second decoder and waits for the rest of the information to catch up. While the second decoder is waiting, the first decoder makes an estimate of the transmitted information, interleaves it to match the format of parity 2, and sends it to the second decoder. The second decoder takes information from both the first decoder and the channel and re-estimates the information. This second estimation is looped back to the first encoder where the process starts again. The iterative process of the Turbo Decoder is illustrated below in Fig. 6.30.

Fig. 6.30 *Iterative Decoding of Turbo Code.*

This cycle will continue until certain conditions are met, such as a certain number of iterations being performed. It is from this iterative process that Turbo Coding gets its name. The decoder circulates estimates of the sent data like a turbo engine circulates air. When the decoder is ready, the estimated information is finally kicked out of the cycle and hard decisions are made in the threshold component. The result is the decoded information sequence.

In the following section, we study two decoding methods for the Turbo Codes, in detail.

6.12 TURBO DECODING

We have seen that the Viterbi Algorithm is used for the decoding of convolutional codes. The Viterbi Algorithm performs a systematic elimination of the paths in the trellis. However, such luck does not exist for Turbo Decoder. The presence of the interleaver complicates the matter immensely. Before the discovery of Turbo Codes, a lot of work was being done in the area of

suboptimal decoding strategies for concatenated codes, involving multiple decoders. The symbol-by-symbol *maximum a posteriori* (MAP) algorithm of Bahl, Cocke, Jelinek and Raviv, published in the *IEEE Transactions on Information Theory* in March 1974 also received some attention. It was this algorithm, which was used by Berrou *et al.* in the iterative decoding of their Turbo Codes. In this section, we shall discuss two methods useful for Turbo Decoding:

(A) The modified Bahl, Cocke, Jelinek and Raviv (BCJR) Algorithm.

(B) The Iterative MAP Decoding.

A. MODIFIED BAHL, COCKE, JELINEK AND RAVIV (BCJR) ALGORITHM

The modified BCJR Decoding Algorithm is a symbol-by-symbol decoder. The decoder decides $u_k = +1$ if

$$P(u_k = +1 \mid \mathbf{y}) > P(u_k = -1 \mid \mathbf{y}), \tag{6.44}$$

and it decides $u_k = -1$ otherwise, where $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ is the noisy received word.

More succinctly, the decision $\hat{u}_k$ is given by

$$\hat{u}_k = \text{sign}\,[L(u_k)] \tag{6.45}$$

where $L(u_k)$ is the **Log *A Posteriori* Probability** (LAPP) **Ratio** defined as

$$L(u_k) = \log\left(\frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})}\right) \tag{6.46}$$

Incorporating the code's trellis, this may be written as

$$L(u_k) = \log\left(\frac{\sum_{S^+} p\,(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})}{\sum_{S^-} p\,(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})}\right), \tag{6.47}$$

where $s_k \in S$ is the state of the encoder at time k, S^+ is the set of ordered pairs (s', s) corresponding to all state transitions $(s_{k-1} = s')$ to $(s_k = s)$ caused by data input $u_k = +1$, and S^- is similarly defined for $u_k = -1$.

Let us define

$$\gamma_k(s', s) = p(s_k = s, y_k | s_{k-1} = s'), \tag{6.48}$$

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \alpha_{k-1}(s')\gamma_k(s', s)}{\sum_{s'}\sum_{s'} \alpha_{k-1}(s')\gamma_k(s' s)} \text{ and} \tag{6.49}$$

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s'} \beta_k(s')\gamma_k(s', s)}{\sum_{s'}\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s', s)} \tag{6.50}$$

with boundary conditions

$$\alpha_0(0) = 1 \text{ and } \alpha_0\ ((s \neq 0) = 0,$$
$$\beta_N(0) = 1 \text{ and } \beta_N(s \neq 0) = 0. \tag{6.51}$$

Then the modified BCJR Algorithm gives the LAPP Ratio in the following form

$$L(u_k) = \log\left(\frac{\sum_{s^+} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{s^-} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s)}\right) \tag{6.52}$$

B. ITERATIVE MAP DECODING

The decoder is shown in Fig. 6.31. D1 and D2 are the two decoders. S is the set of 2^m constituent encoder states. y is the noisy received word. Using Baye's rule we can write $L(u_k)$ as

$$L(u_k) = \log\left(\frac{P(y|u_k = +1)}{P(y|u_k = -1)}\right) + \log\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \tag{6.53}$$

with the second term representing *a priori* information. Since $P(u_k = +1) = P(u_k = -1)$ typically, the *a priori* term is usually zero for conventional decoders. However, for iterative decoders, D1 receives extrinsic or soft information for each u_k from D2 which serves as *a priori* information. Similarly, D2 receives extrinsic information from D1 and the decoding iteration proceeds with the each of the two decoders passing soft information along to the other decoder at each half-iteration except for the first. The idea behind extrinsic information is that D2 provides soft information to D1 for each u_k, using only information not available to D1. D1 does likewise for D2.

Fig. 6.31 *Implementation of the Iterative Turbo Decoder.*

At any given iteration, D1 computes

$$L_1(u_k) = L_c y_k^s + L_{21}^e(u_k) + L_{12}^e(u_k) \tag{6.54}$$

where, the first term is the channel value, $L_c = 4E_c / N_0$ (E_c = energy per channel bit), $L_{21}^e(u_k)$ is extrinsic information passed from D2 to D1, and $L_{12}^e(u_k)$ is the extrinsic information from D1 to D2.

$L_{12}^e(u_k)$ and $L_{21}^e(u_k)$ are calculated as follows

$$L_{12}^e(u_k) = \log\left(\frac{\sum_{s^+} \tilde{\alpha}_{k-1}^{(1)}(s')\gamma_k^e(s',s)\tilde{\beta}_k^{(1)}(s)}{\sum_{s^-} \tilde{\alpha}_{k-1}^{(1)}(s')\gamma_k^e(s',s)\tilde{\beta}_k^{(1)}(s)}\right), \tag{6.55}$$

$$L_{21}^e(u_k) = \log\left(\frac{\sum_{s^+} \tilde{\alpha}_{k-1}^{(2)}(s')\gamma_k^e(s',s)\tilde{\beta}_k^{(2)}(s)}{\sum_{s^-} \tilde{\alpha}_{k-1}^{(2)}(s')\gamma_k^e(s',s)\tilde{\beta}_k^{(2)}(s)}\right), \tag{6.56}$$

where

$$\gamma_k^e(s',s) = \exp\left[\frac{1}{2} L_c \gamma_k^p x_k^p\right], \tag{6.57}$$

$$\gamma_k(s',s) = \exp\left[\frac{1}{2} u_k\left(L_e(u_k) + L_c y_k^s\right)\right] . \gamma_k^e(s',s) \tag{6.58}$$

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \alpha_{k-1}(s')\gamma_k(s's)}{\sum_s \sum_{s'} \alpha_{k-1}(s')\gamma_k(s's)}, \text{ and} \tag{6.59}$$

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s'} \tilde{\beta}_k(s)\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}. \tag{6.60}$$

For the above algorithm, each decoder must have full knowledge of the trellis of the constituent encoders, i.e. each decoder must have a table containing the input bits and parity bits for all possible state transitions s' to s. Also, care should be taken that the last m bits of the N-bit information word to be encoded must force encoder1 to the zero state by the Nth bit.

The complexity of convolutional codes has slowed the development of low-cost Turbo Convolutional Codes (TCC) decoders. On the other hand, another type of turbo code, known as **Turbo Product Code** (TPC), uses block codes, solving multiple steps simultaneously, thereby achieving high data throughput in hardware. We give here a brief introduction to product codes. Let us consider two systematic linear block codes C_1 with parameters (n_1, k_1, d_1) and C_2 with parameters (n_2, k_2, d_2) where n_i, k_i and d_i $(i = 1, 2)$ stand for codeword length, number of information bits and minimum Hamming Distance respectively. The concatenation of two block codes (or product code) $\boldsymbol{P} = \boldsymbol{C_1} * \boldsymbol{C_2}$ is obtained (see Fig. 6.32) by the following steps:

Fig. 6.32 *Example of a Product Code $P = C_1 * C_2$*

(i) placing $(k_1 \times k_2)$ information bits in an array of k_1 rows and k_2 columns,
(ii) coding the k_1 rows using code C_2,
(iii) coding the n_2 columns using code C_1.

The parameters of the product code P are : $n = n_1 * n_2$, $k = k_1 * k_2$, $d = d_1 * d_2$ and the code rate R is given by $R_1 * R_2$ where R_i is the code rate of code C_i. Thus, we can build very long block codes with large minimum Hamming Distance by combining short codes with small minimum Hamming Distance. Given the procedure used to construct the product code, it is clear that the $(n_2 - k_2)$ last columns of the matrix are codewords of C_1. By using the matrix generator, one can show that the last rows of matrix P are codewords of C_2. Hence all the rows of matrix P are codewords of C_1 and all the columns of matrix P are codewords of C_2.

Let us now consider the decoding of the rows and columns of a product code P transmitted ona Gaussian Channel using QPSK signaling. On receiving matrix $\boldsymbol{R}$ corresponding to a transmitted codeword $\boldsymbol{E}$, the first decoder performs the soft decoding of the rows (or columns) of P using as input matrix $\boldsymbol{R}$. Soft Input / Soft Output decoding is performed using the new algorithm proposed by $\boldsymbol{R}$. Pyndiah. By subtracting the soft input from the soft output we obtain the extrinsic information $\boldsymbol{W}(2)$ where index 2 indicates that we are considering the extrinsic information for the second decoding of P which was computed during the first decoding of P. The soft input for the decoding of the columns (or rows) at the second decoding of P is given by

$$\boldsymbol{R}(2) = \boldsymbol{R} + a(2)\boldsymbol{W}(2), \tag{6.61}$$

where $a(2)$ is a scaling factor which takes into account the fact that the standard deviation of samples in matrix $\boldsymbol{R}$ and in matrix $\boldsymbol{W}$ are different. The standard deviation of the extrinsic information is very high in the first decoding steps and decreases as we iterate the decoding. This scaling factor a is also used to reduce the effect of the extrinsic information in the soft

decoder in the first decoding steps when the BER is relatively high. It takes a small value in the first decoding steps and increases as the BER tends to 0. The decoding procedure described above is then generalized by cascading elementary decoders illustrated in Fig. 6.33.

Fig. 6.33 *Block Diagram of Elementary Block Turbo Decoder.*

Let us now, briefly, look at the performance of Turbo Codes and compare it to that of other existing schemes. As shown in Fig. 6.34, Turbo Codes are the best practical codes due to their performance at *low* SNR (at high SNRs, the Reed Solomon Codes outperform Turbo Codes!). It is obvious from the graph that the Recursive Systematic Convolutional (RSC) Turbo Code is the best practical code known so far because it can achieve low BER at low SNR and is the closest to the theoretical maximum of channel performance, the Shannon Limit. The magnitude of how well it performs is determined by the coding gain. It can be recalled that the coding gain is the difference in SNR between a coded channel and an uncoded channel for the same performance (BER). Coding gain can be determined by measuring the distance between the

Fig. 6.34 *Comparison of Different Coding Systems.*

SNR values of any of the coded channels and the uncoded channel at a given BER. For example, the coding gain for the RSC Turbo code, with rate 1/2 at a BER of 10^{-5}, is about 8.5 dB. The physical consequence can be visualized as follows. Consider space communication where the received power follows the inverse square law ($P_R \propto 1/d^2$). This means that the Turbo coded signal can either be received 2.65 ($=\sqrt{7}$) times farther away than the uncoded signal (at the same transmitting power), or it only requires 1/7 the transmitting power (for same transmitting distance). Another way of looking at it is to turn it around and talk about portable device battery lifetimes. For instance, since the RSC Turbo Coded Channel requires only 1/7 the power of the uncoded channel, we can say that a device using a Turbo codec, such as a cell phone, has a battery life 7 times longer than the device without any channel coding.

6.13 CONCLUDING REMARKS

The notion of convolutional codes was first proposed by Elias (1954) and later developed by Wozencraft (1957) and Ash (1963). A class of multiple error correcting convolutional code was suggested by Massey (1963). The study of the algebraic structure of convolutional codes was carried out by Massey (1968) and Forney (1970).

Viterbi Decoding was developed by Andrew J. Viterbi, founder of Qualcomm Corporation. His seminal paper on the technique titled "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," was published in *IEEE Transactions on Information Theory*, Volume IT-13, pages 260–269, in April, 1967. In 1968, Heller showed that the Viterbi Algorithm is practical if the constraint length is not too large.

Turbo Codes represent the next leap forward in error correction. Turbo Codes were introduced in 1993 at the *International Conference on Communications* (ICC) by Berrou, Glavieux and Thitimajshima in their paper "Near-Shannon-Limit Error Correction Coding and Decoding— Turbo-Codes". These codes get their name from the fact that the decoded data are recycled through the decoder several times. The inventors probably found this reminiscent of the way a turbocharger operates. Turbo Codes have been shown to perform within 1 dB of the Shannon Limit at a BER of 10^{-5}. They break a complex decoding problem down into simple steps, where each step is repeated until a solution is reached. The term "Turbo Code" is often used to refer to turbo convolutional codes (TCCs)–one form of Turbo Codes. The symbol-by-symbol *Maximum A Posteriori* (MAP) algorithm of Bahl, Cocke, Jelinek and Raviv, published in 1974 (nineteen years before the introduction of Turbo Codes!), was used by Berrou *et al.* for the iterative decoding of their Turbo Codes. The complexity of convolutional codes has slowed the development of low-cost TCC decoders. On the other hand, another type of Turbo Code, known as Turbo Product Code (TPC), uses block codes, solving multiple steps simultaneously, thereby achieving high data throughput in hardware.

SUMMARY

- An important sub-class of tree codes is called Convolutional Codes. Convolutional Codes make decisions based on past information, i.e. memory is required. A (k_0, n_0) tree code that is linear, time-invariant, and has a finite wordlength $k = (m+1)k_0$ is called an (n, k) Convolutional Code.
- For Convolutional Codes, much smaller blocks of uncoded data of length k_0 are used. These are called Information Frames. These Information Frames are encoded into Codeword Frames of length n_0. The rate of this Tree Code is defined as $R = \frac{k_0}{n_0}$.
- The constraint length of a shift register encoder is defined as the number of symbols it can store in its memory.
- For Convolutional Codes, the Generator Polynomial Matrix of size $k_0 \times n_0$ is given by $G(D) = [g_{ij}(D)]$, where, $g_{ij}(D)$ are the generator polynomials of the code. $g_{ij}(D)$ are obtained simply by tracing the path from input i to output j.
- The Wordlength of a Convolutional Code is given by $k = k_0 \cdot \max_{i,j}[\deg g_{ij}(D) + 1]$, the Blocklength is given by $n = n_0 \cdot \max_{i,j}[\deg g_{ij}(D) + 1]$ and the constraint length is given by
$$v = \sum_{i=1}^{k_0} \max_j [\deg g_{ij}(D)]$$
- The encoding operation can simply be described as vector matrix product, $C(D) = I(D)\, G(D)$, or equivalently, $c_j(D) = \sum_{i=1}^{k_0} i_l(D) g_{l,j}(D)$.
- A parity check matrix $\boldsymbol{H}(D)$ is an $(n_0 - k_0)$ by n_0 matrix of polynomials that satisfies $\boldsymbol{G}(D)\boldsymbol{H}(D)^T = 0$, and the syndrome polynomial vector which is a $(n_0 - k_0)$-component row vector is given by $\boldsymbol{s}(D) = \boldsymbol{v}(D)\, \boldsymbol{H}(D)^T$.
- A systematic encoder for a convolutional code has the generator polynomial matrix of the form $\boldsymbol{G}(D) = [\boldsymbol{I} \mid \boldsymbol{P}(D)]$, where $\boldsymbol{I}$ is a k_0 by k_0 identity matrix and $\boldsymbol{P}(D)$ is a k_0 by $(n_0 - k_0)$ matrix of polynomials. The Parity check polynomial matrix for a Systematic Convolutional Encoder is $\boldsymbol{H}(D) = [-\boldsymbol{P}(D)^T \mid \boldsymbol{I}]$.
- A Convolutional Code whose generator polynomials $g_1(D), g_2(D), \ldots, g_{n0}(D)$ satisfy $GCD[g_1(D), g_2(D), \ldots, g_{n0}(D)] = x^a$, for some a is called a Non-Catastrophic Convolutional Code. Otherwise it is called a Catastrophic Convolutional Code.
- The lth minimum distance d_l^* of a Convolutional Code is equal to the smallest Hamming Distance between any two initial codeword segments l frame long that are not identical in the initial frame. If $l = m + 1$, then this $(m+1)^{\text{th}}$ minimum distance is called the minimum distance of the code and is denoted by d^*, where m is the number of information frames that can be stored in the memory of the encoder. In literature, the minimum distance is also denoted by d_{min}.

- If the lth minimum distance of a Convolutional Code is d_l^*, the code can correct t errors occurring in the first l frames provided, $d_l^* \geq 2t+1$. The free distance of a Convolutional Code is given by $d_{free} = \max_l [d_l]$.
- The Free Length n_{free} of a Convolutional Code is the length of the non-zero segment of a smallest weight convolutional codeword of non zero weight. Thus, $d_l = d_{free}$ if $l = n_{free}$, and $d_l < d_{free}$ if $l < n_{free}$. In literature, n_{free} is also denoted by n_∞.
- Another way to find out the d_{free} of a Convolutional Code is use the concept of a generating function, whose expansion provides all the distance information directly.
- The generator matrix for the Convolutional Code is given by

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \cdots \\ \vdots & \vdots & & & & & & & & \vdots \end{bmatrix}$$

- The Viterbi Decoding Technique is an efficient decoding method for Convolutional Codes. Its computational requirements grow exponentially as a function of the constraint length.
- For rate R and constraint length, let d be the largest integer that satisfies $H\left(\frac{d}{n_0 v}\right) \leq 1 - R$.

  Then, at least one Binary Convolutional Code exists with minimum distance d for which the above inequality holds. Here $H(x)$ is the familiar entropy function for a binary alphabet.
- For a binary code with $R = 1/n_0$ the minimum distance d_{min} satisfies $d_{min} \leq \lfloor (n_0 v + n_0)/2 \rfloor$, where $\lfloor I \rfloor$ denotes the largest integer less than or equal to I.
- An upper bound on d_{free} is given by Heller is $d_{free} = \min_{j \geq 1} \left\lfloor \frac{n_0}{2} \frac{2^j}{2^j - 1}(v + j - 1) \right\rfloor$. To calculate the upper bound, the right hand side should be plotted for different integer values of j. The upper bound is the minimum of this plot.
- For Convolutional Codes, the upper bound on the first error probability can be obtained by $P_e \leq T(D)|_{D = 2\sqrt{p(1-p)}}$ and the bit error probability $P_b \leq \frac{1}{k} \left. \frac{\partial T(D, I)}{\partial I} \right|_{I=1, D=2\sqrt{p(1-p)}}$.
- Turbo codes are actually a quasi mix between Block and Convolutional Codes. Turbo Codes typically use at least two convolutional component encoders and two maximum aposteriori (MAP) algorithm component decoders in the Turbo Codes. Although the encoder determines the capability for the error correction, it is the decoder that determines the actual performance.

It's kind of fun to do the impossible.

Walt Disney (1901–1966)

PROBLEMS

6.1 Design a rate ½ Convolutional encoder with a constraint length $v = 4$ and $d^* = 6$.

(i) Construct the State Diagram for this encoder.

(ii) Construct the Trellis Diagram for this encoder.

(iii) What is the d_{free} for this code?

(iv) Give the Generator Matrix, $\boldsymbol{G}$.

(v) Is this code Non-Catastrophic? Why?

6.2 Design a (12, 3) systematic convolutional encoder with a constraint length $v = 3$ and $d^* \geq 8$.

(i) Construct the Trellis Diagram for this encoder.

(ii) What is the d_{free} for this code?

6.3 Consider the binary encoder shown in Fig. 6.35.

Fig. 6.35

(i) Construct the Trellis Diagram for this encoder.

(ii) Write down the values of k_0, n_0, v, m and R for this encoder. $k_0 = 1$, $n_0 = 3$, $v = 4$. $m = 4$. $R = \frac{1}{3}$.

(iii) What are the values of d^* and d_{free} for this code?

(iv) Give the Generator Polynomial Matrix for this encoder.

$G = [D+1 \quad D^3+D^2 \quad D^4+D^3+D^2]$.

6.4 Consider the binary encoder shown in Fig. 6.36.

Fig. 6.36

(i) Write down the values of k, n, v, m and R for this encoder.
(ii) Give the Generator Polynomial Matrix $G(D)$ for this encoder.
(iii) Give the Generator Matrix G for this encoder.
(iv) Give the Parity Check Matrix H for this encoder.
(v) What are the values of d^*, d_{free} and n_{free} for this code?
(vi) Is this encoder optimal in the sense of the Heller Bound on d_{free}
(vii) Encode the following sequence of bits using this encoder: 101 001 001 010 000.

6.5 Consider a convolutional encoder described by its Generator Polynomial Matrix, defined over $GF(2)$:

$$G(D) = \begin{bmatrix} D & 0 & 1 & D^2 & D+D^2 \\ D^2 & 0 & 0 & 1+D & 0 \\ 1 & 0 & D^2 & 0 & D^2 \end{bmatrix}$$

(i) Draw the circuit realization of this encoder using shift registers. What is the value of v?
(ii) Is this a Catastrophic Code? Why?
(iii) Is this code optimal in the sense of the Heller Bound on d_{free}.

6.6 The Parity Check Matrix of the (12, 9) Wyner-Ash code for $m = 2$ is given as follows.

$$H = \left[\begin{array}{cccc|cccc|cccc|cccc|c} 1&1&1&1 & & & & & & & & & & & & & \\ 1&1&0&0 & 1&1&1&1 & & & & & & & & & \\ 1&0&1&0 & 1&1&0&0 & 1&1&1&1 & & & & & \\ 0&0&0&0 & 1&0&1&0 & 1&1&0&0 & 1&1&1&1 & \cdots \\ 0&0&0&0 & 0&0&0&0 & 1&0&1&0 & 1&1&0&0 & \\ & & \vdots & & & & \vdots & & & & \vdots & & & & \vdots & & \end{array}\right]$$

(i) Determine the Generator Matrix, G.
(ii) Determine the Generator Polynomial Matrix, $G(D)$.
(iii) Give the circuit realization of the (12, 9) Wyner-Ash Convolutional Code.
(viii) What are the values of d^* and d_{free} for this code?

6.7 Consider a Convolutional Encoder defined over $GF(4)$ with the Generator Polynomials

$$g_1(D) = 2D^3 + 3D^2 + 1 \text{ and}$$
$$g_2(D) = D^3 + D + 1.$$

(i) What is the minimum distance of this code?
(ii) Is this code Non-Catastrophic? Why?

6.8 Let the Generator Polynomials of a 1/3 binary Convolutional Encoder be given by

$$g_1(D) = D^3 + D^2 + 1,$$
$$g_2(D) = D^3 + D \text{ and}$$
$$g_3(D) = D^3 + 1.$$

(i) Encode the bit stream: 01100011110101.
(ii) Encode the bit stream: 1010101010 ....
(iii) Decode the received bit stream: 00100110111100011 0011.

6.9. Consider a rate ½ Convolutional Encoder defined over $GF(3)$ with the Generator Polynomials

$$g_1(D) = 2D^3 + 2D^2 + 1 \text{ and}$$
$$g_2(D) = D^3 + D + 2.$$

(i) Show the circuit realization of this encoder.
(ii) What is the minimum distance of this code?
(iii) Encode the following string of symbols using this encoder: 2012111002102.
(iv) Suppose the error vector is given by 0010102000201. Construct the received vector and then decode this received vector using the Viterbi Algorithm.

COMPUTER PROBLEMS

6.10 Write a computer program that determines the Heller Bound on d_{free}, given the values for n_0 and v.

6.11 Write a computer program to exhaustively search for good systematic Convolutional Codes. The program should loop over the parameters k_0, n_0, v, m, etc. and determine the Generator Polynomial Matrix (in octal format) for the best Convolutional Code in its category.

6.12 Write a program that calculates the d^* and d_{free} given the generator polynomial matrix of any convolutional encoder.

6.13 Write a computer program that constructs all possible rate ½ Convolutional Encoder for a given constraint length, v and chooses the best code for a given value of v. Using the program, obtain the following plots:

(i) the minimum distance, d^* versus v, and

(ii) the free distance, d_{free} versus v.

Comment on the error correcting capability of Convolutional Codes in terms of the memory requirement.

6.14 Write a Viterbi Decoder in software that takes in the following:

(i) code parameters in the Octal Format, and

(ii) the received bit stream

The decoder then produces the survivors and the decoded bit stream.

6.15 Verify the Heller Bound on the entries in Table 6.4 for $v = 3, 4, ..., 7$.

6.16 Write a generalized computer program for a Turbo Encoder. The program should take in the parameters for the two encoders and the type of interleaver. It should then generate the encoded bit-stream when an input (uncoded) bit-stream is fed into the program.

6.17 Modify the Turbo Encoder program developed in the previous question to determine the d_{free} of the Turbo Encoder.

6.18 Consider a rate 1/3 Turbo Encoder shown in Fig. 6.37. Let the random interleaver size be 256 bits.

(i) Find the d_{free} of this Turbo encoder.

(ii) If the input bit rate is 28.8 kb/s, what is the time delay caused by the Encoder.

6.19 Write a generalized computer program that performs Turbo Decoding using the iterative MAP Decoding algorithm. The program should take in the parameters for the two encoders, the type of interleaver used for encoding and the SNR. It should produce a sequence of decoded bits when fed with a noisy, encoded bit-stream.

6.20 Consider the rate 1/3 Turbo Encoder comprising the following constituent encoders:

$$G_1(D) = G_2(D) = \left(1 \quad \frac{1 + D^2 + D^3 + D^4}{1 + D + D^4}\right)$$

The encoded output consists of the information bit, followed by the two *parity* bits from the two encoders. Thus the rate of the encoder is 1/3. Use a random interleaver of size 256.

Fig. 6.37 *Turbo Encoder for Problem 6.18.*

(i) For this Turbo Encoder, generate a plot for the bit error rate (BER) versus the signal to noise ratio (SNR). Vary the SNR from –2 dB through 10 dB.

(ii) Repeat the above for an interleaver of size 1024. Comment on your results.

7

Trellis Coded Modulation

A mathematical theory is not to be considered complete until you have made it so clear that you can explain it to the first man whom you meet on the street.

David Hilbert (1862-1943)

7.1 INTRODUCTION TO TCM

In the previous chapters we have studied a number of error control coding techniques. In all these techniques, extra bits are added to the information bits in a *known* manner. However, the improvement in the Bit Error Rate is obtained at the expense of bandwidth caused by these extra bits. This bandwidth expansion is equal to the reciprocal of the code rate.

For example, an RS (255, 223) Code has a code rate $R = 223/255 = 0.8745$ and $1/R = 1.1435$. Hence, to send 100 information bits, we have to transmit 14.35 extra bits (overhead). This translates to a bandwidth expansion of 14.35%. Even for this efficient RS (255, 223) code, the excess bandwidth requirement is not small.

In power limited channels (like deep space communications) one may trade the bandwidth expansion for a desired performance. However, for bandwidth limited channels (like the telephone channel), this may not be the ideal option. In such channels, a bandwidth efficient signalling scheme such as Pulse Amplitude Modulation (PAM), Quadrature Amplitude

Modulation (QAM) or Multi Phase Shift Keying (MPSK) is usually employed to support high bandwidth efficiency (in bit/s/Hz).

In general, either extra bandwidth or a higher signal power is needed in order to improve the performance (error rate). Is it possible to achieve an improvement in system performance *without* sacrificing either the bandwidth (which translates to the data rate) or using additional power? In this chapter we study a coding technique called the **Trellis Coded Modulation Technique**, which can achieve better performance *without* bandwidth expansion or using extra power.

We begin this chapter by introducing the concept of coded modulation. We, then, study some design techniques to construct good Coded Modulation Schemes. Finally, the performance of different Coded Modulation Schemes are discussed for Additive White Gaussian Noise (AWGN) Channels as well as for Fading Channels.

7.2 THE CONCEPT OF CODED MODULATION

Traditionally, coding and modulation were considered two separate parts of a digital communications system. The input message stream is first channel encoded (extra bits are added) and then these encoded bits are converted into an analog waveform by the modulator. The objective of both the channel encoder and the modulator is to correct errors resulting from use of a non-ideal channel. Both these blocks (the encoder and the modulator) are optimized *independently* even though their objective is the same, that is, to correct errors introduced by the channel! As we have seen, a higher performance is possible by lowering the code rate at the cost of bandwidth expansion and increased decoding complexity. However, it is possible to obtain Coding Gain *without* bandwidth expansion if the channel encoder is integrated with the modulator. We illustrate this by a simple example.

Example 7.1 Consider data transmission over a channel with a throughput of 2 bits/s/Hz. One possible solution is to use uncoded QPSK. Another possibility is to first use a rate 2/3 Convolutional Encoder (which converts 2 uncoded bits to 3 coded bits) and then use an 8-PSK signal set which has a throughput of 3 bit/s/Hz. This coded 8-PSK scheme yields the same information data throughput of the uncoded QPSK (2 bit/s/Hz). Note that both the QPSK and the 8-PSK schemes require the same bandwidth. But we know that the symbol error rate for the 8-PSK is worse than that of QPSK for the same energy per symbol. However, the 2/3 convolutional encoder would provide some coding gain. It may be possible that the coding gain provided by the encoder outweighs the performance loss because of the 8-PSK signal set. If the coded modulation scheme performs superior to the uncoded one at the same SNR, we can claim that an improvement is achieved without sacrificing either the data rate or the bandwidth. In this example we have combined a trellis encoder with the modulator. Such a scheme is called a **Trellis Coded Modulation (TCM)** scheme.

We observe that the expansion of the signal set to provide redundancy results in the shrinking of the Euclidean distance between the signal points, if the average signal energy is to be kept constant (Fig. 7.1). This reduction in the Euclidean distance increases the error rate which should be compensated with coding (increase in the Hamming Distance). Here we are assuming an AWGN channel. We also know that the use of a *hard-decision* demodulation prior to decoding in a coded scheme causes an irreversible loss of information. This translates to a loss of SNR. For coded modulation schemes, where the expansion of the signal set implies a power penalty, the use of *soft-decision* decoding is imperative. As a result, demodulation and decoding should be combined in a single step, and the decoder should operate on the *soft* output samples of the channel. For maximum likelihood decoding using soft-decisions, the optimal decoder chooses that code sequence which is nearest to the received sequence in terms of the Euclidean distance. Hence, an efficient coding scheme should be designed based on maximizing the minimum Euclidean distance between the coded sequences rather than the Hamming Distance.

Fig. 7.1 *The Euclidean Distances between the Signal Points for QPSK and 8-PSK.*

The Viterbi Algorithm can be used to decode the received symbols for a TCM scheme. In the previous chapter we saw that the basic idea in Viterbi decoding is to trace out the most likely path through the trellis. The most likely path is that which is closest to the received sequence in terms of the Hamming Distance. For a TCM scheme, the Viterbi decoder chooses the most likely path in terms of Euclidean Distance. The performance of the decoding algorithm depends on the minimum Euclidean distance between a pair of paths forming an error event.

Definition 7.1 The minimum Euclidean Distance between any two paths in the trellis is called the **Free Euclidean Distance**, d_{free} of the TCM scheme.

In the previous chapter we had defined d_{free} in terms of Hamming Distance between any two paths in the trellis. The minimum free distance in terms of Hamming Weight could be calculated as the minimum weight of a path that deviates from the all zero path and later merges back into the all zero path at some point further down the trellis. This was a consequence of the linearity of Convolutional Codes. However, the same does not apply for the case of TCM, which is non linear. It may be possible that d_{free} is the Euclidean Distance between two paths in the trellis neither of which is the all zero path. Thus, in order to calculate the Free Euclidean Distance for a TCM scheme, all pairs of paths have to be evaluated.

Example 7.2 Consider the convolutional encoder followed by a modulation block performing natural mapping ($000 \rightarrow s_0$, $001 \rightarrow s_1$, ..., $111 \rightarrow s_7$) shown in Fig. 7.2. The rate of the encoder is 2/3. It takes in two bits at a time (a_1, a_2) and outputs three encoded bits (c_1, c_2, c_3). The three output bits are then mapped to one of the eight possible signals in the 8-PSK signal set.

Fig. 7.2 *The TCM Scheme for Example 7.2.*

This combined encoding and modulation can also be represented using a trellis with its branches labelled with the output symbol s_i. The TCM scheme is depicted below. This is a fully connected trellis. Each branch is labelled by a symbol from the 8-PSK constellation diagram. In order to represent the symbol allocation unambiguously, the assigned symbols to the branches are written at the front end of the trellis. The convention is as follows. Consider state 1. The branch from state 1 to state 1 is labelled with s_0, branch from state 1 to state 2 is labelled with s_7, branch from state 1 to state 3 is labelled with s_5 and branch from state 1 to state 4 is labelled with s_2. So, the 4-tuple $\{s_0, s_7, s_5, s_2\}$ in front of state 1 represents the branch labels emanating from state 1 in sequence. To encode any incoming bit stream, we follow the same procedure as for convolutional encoder. However, in the case of TCM, the output is a sequence of symbols rather than a sequence of bits. Suppose we have to encode the bit stream 1 0 1 1 1 0 0 0 1 0 0 1 ... We first group the input sequence in pairs because the input is two bits at a time. The grouped input sequence is

10 11 10 00 ...

The TCM encoder output can be obtained simply by following the path in the trellis as dictated by the input sequence. The first input pair is 10. Starting from the first node in state 0, we traverse the third branch emanating from this node as dictated by the input 01. This takes us to state 2. The

symbol output for this branch is s_5. From state 2 we move along the fourth branch as determined by the next input pair 11. The symbol output for this branch is s_1. In this manner, the output symbols corresponding to the given input sequence is

$$s_5\,,\ s_1,\ s_3,\ s_3\ \ldots$$

Fig. 7.3 *The Path in the Trellis Corresponding to the Input Sequence 10 11 10 00 ...*

The path in the Trellis Diagram is depicted by the bold lines in Fig. 7.3. As in the case of convolutional encoder, in TCM too, every encoded sequence corresponds to a unique path in the trellis. The objective of the decoder is to recover this path from the Trellis Diagram.

Example 7.3 Consider the TCM scheme shown in Example 7.2. The free Euclidean Distance, d_{free} of the TCM scheme can be found by inspecting all possible pairs of paths in the trellis. The two paths that are separated by the minimum squared Euclidean Distance (which yields the d_{free}) are shown in the Trellis Diagram given in Fig. 7.4 with bold lines.

Fig. 7.4 *The Two Paths in the Trellis that have the Free Euclidean Distance, d^2_{free}*

$$d^2_{free} = d^2_E(s_0, s_7) + d^2_E(s_0, s_0) + d^2_E(s_2, s_1)$$
$$= \delta_0^2 + 0 + \delta_0^2 = 2\delta_0^2 = 1.172\ E_s$$

It can be seen that in this case, the error event that results in d_{free} does not involve the all zero sequence. As mentioned before, in order to find the d_{free}, we must evaluate *all* possible pairs of paths in the trellis. It is not sufficient just to evaluate the paths diverging from and later merging back into the all zero path because of the non-linear nature of TCM.

We must now develop a method to compare the coded scheme with the uncoded one. We introduce the concept of coding gain below.

Definition 7.2 The difference between the values of the *SNR* for the coded and uncoded schemes required to achieve the same error probability is defined as the *Coding Gain, g.*

$$g = \text{SNR}|_{uncoded} - \text{SNR}|_{coded} \tag{7.1}$$

At high *SNR*, the coding gain can be expressed as

$$g_\infty = g|_{SNR\to\infty} = 10\log\frac{\left(d^2_{free}/E_s\right)_{coded}}{\left(d^2_{free}/E_s\right)_{uncoded}} \tag{7.2}$$

where g_∞ represents the **Asymptotic Coding Gain** and E_s is the average signal energy. For uncoded schemes, d_{free} is simply the minimum Euclidean Distance between the signal points.

Example 7.4 Consider the TCM scheme discussed in Example 7.2 in which the encoder takes in 2 bits at a time. If we were to send uncoded bits, we would employ QPSK. The df_{ree} for the uncoded scheme (QPSK) is $2E_s$ from Fig. 7.1. From Example 7.3 we have $d_{free} = 1.172E_s$ for our TCM scheme. The Asymptotic Coding Gain is then given by

$$g_\infty = 10\log\frac{1.172}{2} = -2.3\ \text{dB}.$$

This implies that the performance of our TCM scheme is actually *worse* than the uncoded scheme. A quick look at the convolutional encoder used in this example suggests that it has good properties in terms of Hamming Distance. In fact, it can be verified that this convolutional encoder is optimal in the sense of maximizing the free Hamming Distance. However, the encoder fails to perform well for the case of TCM. This illustrates the point that TCM schemes must be designed to maximize the Euclidean Distance rather than the Hamming Distance.

For a fully connected trellis discussed in Example 7.2, by a proper choice of the mapping scheme, we can improve the performance. In order to design a better TCM scheme, it is possible to directly work from the trellis onwards. The objective is to assign the 8 symbols from the 8-PSK signal set in such a manner that the d_{free} is maximized. One approach is to use an exhaustive computer search. There are a total of 16 branches that have to be assigned labels (symbols) from time t_k to t_{k+1}. We have 8 symbols to choose from. Thus, an exhaustive search would involve 8^{16} different cases!

Another approach is to assign the symbols to the branches in the trellis in a heuristic manner so as to increase the d_{free}. We know that an error event consists of a path diverging in one state and then merging back after one or more transitions, as depicted in Fig. 7.5. The Euclidean Distance associated with such an error event can be expressed as

$$d^2_{total} = d^2_E \text{ (diverging pair of paths)} + \ldots + d^2_E \text{ (re-merging pair of paths)} \tag{7.3}$$

Fig. 7.5 *An Error Event.*

Thus, in order to design a TCM scheme with a large d_{free}, we can at least ensure that the d^2_E (diverging pair of paths) and the d^2_E (re-merging pair of paths) are as large as possible. In TCM schemes, a redundant 2^{m+1}-ary signal set is often used to transmit m bits in each signalling interval. The m input bits are first encoded by an $m/(m+1)$ convolutional encoder. The resulting $m+1$ output bits are then mapped to the signal points of the 2^{m+1}-ary signal set. Now, recall that the maximum likelihood decoding rule for the AWGN channel with soft decision decoding is to *minimize the squared Euclidean Distance* between the received vector and the code vector estimate from the trellis diagram (see Section 6.7, Chapter 6). Therefore, the mapping is done in such a manner as to maximize the minimum Euclidean Distance between the different paths in the trellis. This is done using a rule called Mapping by **Set Partitioning**.

7.3 MAPPING BY SET PARTITIONING

The Mapping by Set Partitioning is based on successive partitioning of the expanded 2^{m+1}-ary signal set into subsets with increasing minimum Euclidean Distances. Each time we partition the set, we reduce the number of the signal points in the subset, but increase the minimum distance between the signal points in the subset. The set partitioning can be understood by the following example.

Example 7.5 Consider the set partitioning of 8-PSK. Before partitioning, the minimum Euclidean Distance of the signal set is $\Delta_0 = \delta_0$. In the first step, the 8 points in the constellation diagram are subdivided into two subsets, A_0 and A_1, each containing 4 signal points as shown in Fig. 7.6. As a result of this first step, the minimum Euclidean Distance of each of the subsets is now $\Delta_0 = \delta_1$, which is larger than the minimum Euclidean Distance of the original 8-PSK. We continue this procedure and subdivide the sets A_0 and A_1 and into two subsets each, $A_0 \rightarrow \{A_{00}, A_{01}\}$ and $A_1 \rightarrow \{A_{10}, A_{11}\}$. As a result of this second step, the minimum Euclidean Distance of each of the subsets is now $\Delta_2 = \delta_3$. Further subdivision results in one signal point per subset.

Fig. 7.6 *Set Partitioning of the 8-PSK S•signal Set.*

Consider the expanded 2^{m+1}-ary signal set used for TCM. In general, it is not necessary to continue the process of set partitioning until the last stage. The partitioning can be stopped as soon as the minimum distance of a subset is larger than the desired minimum Euclidean Distance of the TCM scheme to be designed. Suppose the desired Euclidean Distance is obtained just after the $\tilde{m}+1^{th}$ set partitioning step ($\tilde{m} \le m$). It can be seen that after $\tilde{m}+1$ steps we have $2^{\tilde{m}+1}$ subsets and each subset contains $2^{m-\tilde{m}}$ signal points.

A general structure of a TCM encoder is given in Fig. 7.7. It consists of m input bits of which the $\tilde{m}$ bits are fed into a rate $\tilde{m}/(\tilde{m}+1)$ convolutional encoder while the remaining $m-\tilde{m}$ bits are left uncoded. The $\tilde{m}+1$ output bits of the encoder along with the $m-\tilde{m}$ uncoded bits are then input to the signal mapper. The signal mapper uses the $\tilde{m}+1$ bits from the convolutional encoder to select one of the possible $2^{\tilde{m}+1}$ subsets. The remaining $m-\tilde{m}$ uncoded bits are used to select one of the $2^{\tilde{m}+m}$ signals from this subset. Thus, the input to the TCM encoder are m bits and the output is a signal point chosen from the original constellation.

Fig. 7.7 *The General Structure of a TCM Encoder.*

For the TCM encoder shown in Fig. 7.7 we observe that $m - \tilde{m}$ uncoded bits have no effect on the state of the convolutional encoder because the input is not being altered. Thus, we can change the first $m - \tilde{m}$ bits of the total m input bits *without* changing the encoder state. This implies that $2^{m-\tilde{m}}$ **parallel transitions** exist between states. These parallel transitions are associated with the signals of the subsets in the lowest layer of the set partitioning tree. For the case of $m = \tilde{m}$, the states are joined by single transitions.

Let us denote the minimum Euclidean Distance between parallel transitions by $\Delta_{\tilde{m}+1}$ and the minimum Euclidean Distance between non-parallel paths of the trellis by $d_{free}(\tilde{m})$. The free Euclidean Distance of the TCM encoder shown in Fig. 7.7 can then be written as

$$d_{free} = \min\,[\Delta_{\tilde{m}+1},\ d_{free}(\tilde{m})]. \tag{7.4}$$

Example 7.6 Consider the TCM scheme proposed by Ungerboeck. It is designed to maximize the Free Euclidean Distance between coded sequences. It consists of a rate 2/3 convolutional encoder coupled with an 8-PSK signal set mapping. The encoder is given in Fig. 7.8 and the corresponding trellis diagram in Fig. 7.9.

Fig. 7.8 *The TCM Encoder for Example 7.6.*

Fig. 7.9 *The Trellis Diagram for the Encoder in Example 7.6.*

For this encoder $m = 2$ and $\tilde{m} = 1$, which implies that there are $2^{m-\tilde{m}} = 2^1 = 2$ parallel transitions between each state. The minimum squared Euclidean distance between parallel transitions is

$$\Delta^2_{m+1} = \Delta_2^2 = \delta_2^2 = 4E_s\,.$$

The minimum squared Euclidean Distance between non-parallel paths in the trellis, $d_{free}(\tilde{m})$, is given by the error event shown in Fig. 7.9 by bold lines. From the figure, we have

$$\begin{aligned} d_{free}^2(\tilde{m}) &= d_E^2(s_0, s_2) + d_E^2(s_0, s_1) + d_E^2(s_0, s_2) \\ &= \delta_1^2 + \delta_0^2 + \delta_1^2 = 4.586\ E_s\,. \end{aligned}$$

The error events associated with the parallel paths have the minimum squared Euclidean Distance among all the possible error events. Therefore, the minimum squared Euclidean Distance for the TCM scheme is $d_{free}^2 = \min\left[\Delta^2_{\tilde{m}+1}, d^2_{free}(m)\right] = 4E_s$. The asymptotic coding gain of this scheme is

$$g_\infty = 10 \log \frac{4}{2} = 3 \text{ dB}$$

This shows that the TCM scheme proposed by Ungerboeck shows an improvement of 3 dB over the uncoded QPSK. This example illustrates the point that the combined coded modulation scheme can compensate for the loss from the expansion of the signal set by the coding gain achieved by the convolutional encoder. Further, for the non-parallel paths

$$\begin{aligned} d_{total}^2 &= d_E^2\,(\text{diverging pair of paths}) + \ldots + d_E^2\,(\text{re-merging pair of paths}) \\ &= \delta_1^2 + \ldots + \delta_1^2 = (\delta_1^2 + \delta_1^2) + \ldots = \delta_3^2 + \ldots = 4E_s + \ldots \end{aligned}$$

However, the minimum squared Euclidean Distance for the parallel transition is $\delta_3^2 = 4E_s$. Hence, the minimum squared Euclidean Distance of this TCM scheme is determined by the parallel transitions.

7.4 UNGERBOECK'S TCM DESIGN RULES

In 1982 Ungerboeck proposed a set of design rules for maximizing the free Euclidean Distance for TCM schemes. These design rules are based on heuristics.

Rule 1: **Parallel Transitions**, if present, must be associated with the signals of the subsets in the lowest layer of the **Set Partitioning Tree**. These signals have the minimum Euclidean Distance $\Delta_{\tilde{m}+1}$.

Rule 2: The transitions originating from or merging into one state must be associated with signals of the first step of set partitioning. The Euclidean Distance between these signals is at least Δ_1.

Rule 3: All signals are used with equal frequency in the Trellis Diagram.

Example 7.7 Next, we wish to improve upon the TCM scheme proposed in Example 7.6. We observed in the previous example that the parallel transitions limit the d^2_{free}. Therefore, we must come up with a trellis that has no parallel transitions. The absence of parallel paths would imply that the d^2_{free} is not limited to d_3^2, the maximum possible separation between two signal points in the 8-PSK Constellation Diagram. Consider the Trellis Diagram shown in Fig. 7.10. The trellis has 8 states. There are no Parallel Transitions in the Trellis Diagram. We wish to assign the symbols from an 8-PSK signal set to the branches of this trellis according to the Ungerboeck rules.

Since there are no parallel transitions here, we start directly with Ungerboeck's second rule. We must assign the transitions originating from or merging into one state with signals from the first step of set partitioning. We will refer to Fig. 7.6 for the Set Partitioning Diagram for 8-PSK. The first step of set partitioning yields two subsets, A_0 and A_1, each consisting of four signal points. We first focus on the diverging paths. Consider the topmost node (state S_0). We assign to these four diverging paths the signals s_0, s_4, s_2 and s_6. Note that they all belong to the subset A_0. For the next node (state S_1), we assign the signals s_1, s_5, s_3 and s_7 belonging to the subset A_1. For the next node (state S_2), we assign the signals s_4, s_0, s_6 and s_2 belonging to the subset A_0. The order has been shuffled to ensure that at the re-merging end we still have signals from the first step of set partitioning. If we observe the four paths that merge into the node of state S_0, their branches are labelled s_0, s_4, s_2 and s_6, which belong to A_0. This clever assignment has ensured that the transitions originating from or merging into one state are labelled with signals from the first step of set partitioning, thus satisfying rule 2. It can be verified that all the signals have been used with equal frequency. We did not have to make any special effort to ensure that.

The error event corresponding to the squared free Euclidean Distance is shown in the Trellis Diagram with bold lines. The squared free Euclidean Distance of this TCM Scheme is

$$d^2_{free} = d^2_E(s_0, s_6) + d^2_E(s_0, s_7) + d^2_E(s_0, s_6)$$
$$= \delta_1^2 + \delta_0^2 + \delta_1^2 = 4.586\ E_s$$

Fig. 7.10 *The Trellis Diagram for the Encoder in Example 7.7.*

In comparison to uncoded QPSK, this translates to an asymptotic coding gain of

$$g_\infty = 10 \log \frac{4.586}{2} = 3.60 \text{ dB}$$

Thus, at the cost of added encoding and decoding complexity, we have achieved a 0.6 dB gain over the TCM scheme discussed in Example 7.6.

Example 7.8 Consider the 8 state, 8-PSK TCM scheme discussed in Example 7.7. The equivalent systematic encoder realization with feedback is given in Fig. 7.11.

Fig. 7.11 *The TCM Encoder for Example 7.7.*

Let us represent the output of the convolutional encoder shown in Fig. 7.11 in terms of the input and delayed versions of the input (See Section 6.3 of Chapter 6 for analytical representation of Convolutional Codes). From the figure, we have

$$c_1(D) = a_1(D),$$
$$c_2(D) = a_2(D),$$

$$c_3(D) = \left(\frac{D^2}{1+D^2}\right) a_1(D) + \left(\frac{D^2}{1+D^3}\right) a_2(D)$$

Therefore, the Generator Polynomial Matrix of this encoder is

$$G(D) = \begin{bmatrix} 1 & 0 & \dfrac{D^2}{1+D^3} \\ 0 & 1 & \dfrac{D}{1+D^3} \end{bmatrix}$$

and the parity check polynomial matrix, $\boldsymbol{H}(D)$, satisfying $\boldsymbol{G}(D).\ \boldsymbol{H}^T(D) = \mathbf{0}$ is

$$\boldsymbol{H}(D) = [D^2\ D_1 + D^3\ ].$$

We can re-write the parity check polynomial matrix $\boldsymbol{H}(D) = [H_1(D)\ H_2(D)\ H_3(D)]$, where

$$H_1(D) = D^2 = (000\ 100)_{\text{binary}} = (04)_{\text{octal}},$$
$$H_2(D) = D = (000\ 010)_{\text{binary}} = (02)_{\text{octal}},$$
$$H_3(D) = 1+D^3 = (001\ 001)_{\text{binary}} = (11)_{\text{octal}}.$$

Table 7.1 gives the encoder realization and asymptotic coding gains of some of the good TCM Codes constructed for the 8-PSK signal constellation. Almost all of these TCM schemes have been found by exhaustive computer searches. The coding gain is given with respect to the uncoded QPSK. The parity check polynomials are expressed octal form.

Table 7.1 *TCM Schemes Using 8-PSK*

No. of states	H_1	H_2	H_3	d^2_{free}/E_s	g_a (dB)
4	–	2	5	4.00	3.01
8	04	02	11	4.58	3.6
16	16	04	23	5.17	4.13
32	34	16	45	5.75	4.59
64	066	030	103	6.34	5.01
128	122	054	277	6.58	5.17
256	130	072	435	7.51	5.75

Example 7.9 We now look at a TCM scheme that involves 16QAM. The TCM encoder takes in 3 bits and outputs one symbol from the 16QAM Constellation Diagram. This TCM scheme has a throughput of 3 bits/s/Hz and we will compare it with uncoded 8-PSK, which also has a throughput of 3 bits/s/Hz.

Let the minimum distance between two points in the Signal Constellation of 16QAM be δ_0 as depicted in Fig. 7.12. It is assumed that all the signals are equiprobable. Then the average signal energy of a 16QAM signal is obtained as

Fig. 7.12 *Set Partitioning of 16QAM.*

$$E_s = \frac{1}{16}(2\delta_0^2 + 10\delta_0^2 + 10\delta_0^2 + 18\delta_0^2) = \frac{10}{4}\delta_0^2$$

Thus we have,

$$\delta_0 = 2\sqrt{\frac{E_s}{10}}$$

The Trellis Diagram for the 16QAM TCM scheme is given in Fig. 7.13. The trellis has 8 states. Each node has 8 branches emanating from it because the encoder takes in 3 input bits at a time ($2^3 = 8$). The encoder realization is given in Fig. 7.14. The Ungerboeck design rules are followed to assign the symbols to the different branches. The branches diverging from a node and the branches merging back to a node are assigned symbols from the set A_0 and A_1. The parallel paths are assigned symbols from the lowest layer of the Set Partitioning Tree (A_{000}, A_{001}, etc.).

The squared Euclidean Distance between any two parallel paths is $\Delta_3^2 = 8\delta_0^2$. This is by design as we have assigned symbols to the parallel paths from the lowest layer of the set Partitioning Tree. The minimum squared Euclidean Distance between non-parallel paths is

$$d_E^2 = \Delta_1^2 + \Delta_0^2 + \Delta_1^2 = 5\delta_0^2$$

Therefore, the free Euclidean Distance for the TCM scheme is

$$d_{free}^2 = \min\ [8\delta_0^2, 5\delta_0^2] = 5\delta_0^2 = 2E_s.$$

Note that the free Euclidean Distance is determined by the non-parallel paths rather than the parallel paths. We now compare the TCM scheme with the uncoded 8-PSK, which has the same throughput. For uncoded 8-PSK, the minimum squared Euclidean Distance is $(2-\sqrt{2})E_s$. Thus, the asymptotic coding gain for this TCM encoder is

$$g_\infty = 10\log\frac{2}{2-\sqrt{2}} = 5.3 \text{ dB}$$

Fig. 7.13 *Trellis Diagram for the 16 QAM TCM Scheme.*

Fig. 7.14 *The Equivalent Systematic Encoder for the Trellis Diagram Shown in Fig. 7.13.*

7.5 TCM DECODER

We have seen that, like Convolutional Codes, TCM Schemes are also described using Trellis Diagrams. Any input sequence to a TCM encoder gets encoded into a sequence of symbols based on the Trellis Diagram. The encoded sequence corresponds to a particular path in this trellis diagram. There exists a one-to-one correspondence between an encoded sequence and a path within the trellis. The task of the TCM decoder is simply to identify the most likely path in the trellis. This is based on the maximum likelihood criterion. As seen in the previous chapter, an efficient search method is to use the Viterbi algorithm (see Section 6.7 of the previous chapter).

For soft decision decoding of the received sequences using the Viterbi Algorithm, each trellis branch is labelled by the branch metric based on the observed received sequence. Using the maximum likelihood decoder for the Additive White Gaussian Noise (AWGN) channels, the branch metric is defined as the Euclidean Distance between the coded sequence and the received sequence. The Viterbi Decoder finds a path through the trellis which is closest to the received sequence in the Euclidean Distance sense.

Definition 7.3 The **branch metric** for a TCM scheme designed for AWGN channel is the *Euclidean Distance* between the received signal and the signal associated with the corresponding branch in the trellis.

In the next section, we study the performance of TCM schemes over AWGN channels. We also develop some design rules.

7.6 PERFORMANCE EVALUATION FOR AWGN CHANNEL

There are different performance measures for a TCM scheme designed for an AWGN channel. We have already discussed the asymptotic coding gain, which is based on free Euclidean Distance, d_{free}. We will now look at some other parameters that are used to characterize a TCM Code.

Definition 7.4 The **average number of nearest neighbours** at free distance, $N(d_{free})$, gives the average number of paths in the trellis with free Euclidean Distance d_{free} from a transmitted sequence. This number is used in conjunction with d_{free} for the evaluation of the error event probability.

Definition 7.5 Two finite length paths in the trellis form an **error event** if they start form the same state, diverge and then later merge back. An error event of length l is defined by two coded sequences s_n and $\tilde{s}_n$,

$$\mathbf{s}_n = (s_n, s_{n+1}, \ldots, s_{n+l+1})$$

$$\tilde{\mathbf{s}}_n = (\hat{s}_n, \hat{s}_{n+1}, \cdots, \hat{s}_{n+l+1}),$$

such that

$$s_n = \hat{s}_n$$

$$s_{n+l+1} = \hat{s}_{n+l+1}$$

$$s_i \neq \hat{s}_i,\ i = n+1, \ldots, n+l \tag{7.5}$$

Definition 7.6 The probability of an error event starting at time n, given that the decoder has estimated the correct transmitter state at that time, is called the **error event probability**, P_e.

The performance of TCM schemes is generally evaluated by means of upper bounds on error event probability. It is based on the generating function approach. Let us consider again the Ungerboek model for rate $m/(m+1)$ TCM scheme as shown in Fig. 7.7. The encoder takes in m bits at a time and encodes it to $m+1$ bits, which are then mapped by a memoryless mapper, $f(.)$, on to a symbol s_i. Let us call the binary $(m+1)$-tuples c_i as the labels of the signals s_i. We observe that there is a one-to-one correspondence between a symbol and its label. Hence, an error event of length l can be equivalently described by two sequences of labels

$$C_l = (c_k, c_{k+1}, \ldots, c_{k+l-1}) \text{ and } C'_l = (c'_k, c'_{k+1}, \ldots, c'_{k+l-1}), \tag{7.6}$$

where, $c'_k = c_k \oplus e_k$, $c'_{k+1} = c_{k+1} \oplus e_{k+1}$, ..., and $E_l = (e_k, e_{k+1}, \ldots, e_{k+l-1})$ is a sequence of binary error vectors. The mathematical symbol $\oplus$ represents binary (modulo-2) addition. An error event of length l occurs when the decoder chooses, instead of the transmitted sequence C_l, the sequence C'_l which corresponds to a path in the trellis diagram that diverges from the original transmitted path and re-merges back exactly after l time intervals. To find the probability of error, we need to sum over all possible values of l the probabilities of error events of length l (i.e., joint probabilities that C_l is transmitted and C'_l is detected). The upper bound on the probability of error is obtained by the following union bound

$$P_e \le \sum_{l=1}^{\infty} \sum_{s_l} \sum_{s'_l \ne s_l} P(s_l) P(s_l, s'_l) \tag{7.7}$$

where $P(s_l\, s'_l)$ denotes the pairwise error probability (i.e., probability that the sequence s_l is transmitted and the sequence s'_l is detected). Assuming a one-to-one correspondence between a symbol and its label, we can write

$$P_e \le \sum_{l=1}^{\infty} \sum_{C_l} \sum_{C'_l \ne C_l} P(C_l) P(C_l, C'_l)$$

$$= \sum_{l=1}^{\infty} \sum_{C_l} \sum_{E_l \ne 0} P(C_l) P(C_l \oplus E_l) \tag{7.8}$$

The pairwise error probability $P(C_l, C_l, E_l)$ can be upper-bounded by the Bhattacharyya Bound (see Problem 7.12) as follows

$$P(C_l, C_l, \oplus E_l) \le e^{-\left\{\frac{1}{4N_0}\|f(C_l) - f(C'_l)\|^2\right\}}$$

$$= e^{-\left\{\frac{1}{4N_0}\sum_{n=1}^{l}\|f(C_l) - f(C'_l)\|^2\right\}} \tag{7.9}$$

where $f(.)$ is the memoryless mapper. Let $D = e^{-\left\{\frac{1}{4N_0}\right\}}$ (for Additive White Gaussian Noise Channel with single sided power spectral density N_0), then

$$P(C_b\, C_b \oplus E_l) \le D^{\|f(C_l) - f(C'_l)\|^2} = D^{d_E^2\,(f(C_l), f(C'_l))} \tag{7.10}$$

where $d_E^2(f(C_l), f(C'_l))$ represents the squared Euclidean distance between the symbol sequences s_l and s'_l. Next, define the function

$$W(E_l) = \sum_{C_l} P(C_l)\, D^{\|f(C_l) - f(C_l \oplus E_l)\|^2} \tag{7.11}$$

We can now write the probability of error as

$$P_e \le \sum_{l=1}^{\infty} \sum_{E_l \ne 0} W(E_l) \tag{7.12}$$

From the above equation we observe that the probability of error is upper-bounded by a sum over all possible error events, E_l. Note that

$$d_E^2(f(C_l), f(C_l \oplus E_l)) = \sum_{i=1}^{l} d_E^2(f(c_i), f(c_i \oplus e_i)). \tag{7.13}$$

We now introduce the concept of an error state diagram which is essentially a graph whose branches have matrix labels. We assume that the source symbols are equally probable with probabilities $2^{-m} = 1/M$.

Definition 7.7 The error weight matrix, $G(e_i)$ is an $N \times N$ matrix whose element in the pth row and q^{th} column is defined as

$$[G(e_i)]_{p,q} = \frac{1}{M} \sum_{c_{p\to q}} D^{\|f(c_{p\to q}) - f(c_{p\to q} \oplus e_i)\|^2}, \text{ if there is a transition from state } p \text{ to state } q,$$

$$\text{and } [G(e_i)]_{p,q} = 0, \text{ if there is no transition from state } p \text{ to state } q \text{ of the trellis,} \tag{7.14}$$

where $c_{p\to q}$ are the label vectors generated by the transition from state p to state q.

The summation accounts for the possible parallel transitions (parallel paths) between states in the Trellis Diagram. The entry (p, q) in the matrix G provides an upperbound on the probability that an error event occurs starting from the node p and ending on q. Similarly, $(1/N)G\mathbf{1}$ is a vector whose p^{th} entry is a bound on the probability of any error event starting from node p. Now, to any sequence $E_l = e_1, e_2, \ldots, e_l$, there corresponds a sequence of l error weight matrices $G(e_1), G(e_1), \ldots, G(e_l)$. Thus, we have

$$W(E_l) = \frac{1}{N} \mathbf{1}^T \prod_{n=1}^{l} G(\mathbf{e}_n)\mathbf{1} \tag{7.15}$$

where $\mathbf{1}$ is a column N-vector all elements of which are unity. We make the following observations:

(i) For any matrix $\boldsymbol{A}$, $\mathbf{1}^T \boldsymbol{A}\, \mathbf{1}$ represents the sum of all entries of $\boldsymbol{A}$.

(ii) The element (p, q) of the matrix $= \prod_{n=1}^{l} \boldsymbol{G}(\boldsymbol{e}_n)$ enumerates the Euclidean Distance involved in transition from state p to state q in exactly l steps.

Our next job is to relate the above analysis to the probability of error, P_e. It should be noted that the error vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_l$ are not independent. The Error State Diagram has a structure determined only by the Linear Convolutional Code and differs from the Code State Diagram only in the denomination of its state and branch labels ($\boldsymbol{G}(\boldsymbol{e}_i)$). Since the error vectors $\boldsymbol{e}_i$ are simply the differences of the vectors $\boldsymbol{c}_i$, the connections among the vectors $\boldsymbol{e}_i$ are the same as that among the vectors $\boldsymbol{c}_i$. Therefore, from (7.12) and (7.15) we have

$$P_e \leq T(D)\big|_{D = e^{-1/4N_0}}, \tag{7.16}$$

where

$$T(D) = \frac{1}{N} 1^T \boldsymbol{G}\mathbf{1}, \tag{7.17}$$

and the matrix

$$\boldsymbol{G} = \sum_{l=1}^{\infty} \sum_{E_l \neq 0} \prod_{n=1}^{l} \boldsymbol{G}(\boldsymbol{e}_n) \tag{7.18}$$

is the matrix transfer function of the error state diagram. $T(D)$ is called the scalar transfer function or simply the **transfer function** of the Error State Diagram.

Example 7.10 Consider a rate ½ TCM scheme with $m = 1$, and $M = 4$. It takes one bit at a time and encodes it into two bits, which are then mapped to one of the four QPSK symbols. The two state Trellis Diagram and the symbol allocation from the 4-PSKConstellation is given in Fig. 7.15.

Fig. 7.15

Let us denote the error vector by $\boldsymbol{e} = (\boldsymbol{e}_2\, \boldsymbol{e}_1)$. Then, from (7.14)

$$\boldsymbol{G}(e_2 e_1) = \frac{1}{2}\begin{bmatrix} D^{\|f(00)-f(00\oplus e_2 e_1)\|^2} & D^{\|f(10)-f(10\oplus e_2 e_1)\|^2} \\ D^{\|f(00)-f(01\oplus e_2 e_1)\|^2} & D^{\|f(11)-f(11\oplus e_2 e_1)\|^2} \end{bmatrix} \tag{7.19}$$

$$= \frac{1}{2}\begin{bmatrix} D^{\|f(00)-f(e_2 e_1)\|^2} & D^{\|f(10)-f(\bar{e}_2 e_1)\|^2} \\ D^{\|f(01)-f(e_2 \bar{e}_1)\|^2} & D^{\|f(11)-f(\bar{e}_2 \bar{e}_1)\|^2} \end{bmatrix}$$

where $\bar{e} = 1 \oplus e$. The error state diagram for this TCM scheme is given in Fig. 7.16.

Fig. 7.16

The matrix transfer function of the error state diagram is

$$\boldsymbol{G} = \boldsymbol{G}(10)[\boldsymbol{I}_2 - \boldsymbol{G}(11)]^{-1}\, \boldsymbol{G}(01) \tag{7.20}$$

where $\boldsymbol{I}_2$ is the 2×2 identity matrix. In this case there are only three error vectors possible, {01,10,11}. From (7.19) we calculate

$$\boldsymbol{G}(01) = \frac{1}{2}\begin{bmatrix} D^2 & D^2 \\ D^2 & D^2 \end{bmatrix},\ \boldsymbol{G}(10) = \frac{1}{2}\begin{bmatrix} D^4 & D^4 \\ D^4 & D^4 \end{bmatrix},\ \text{and } \boldsymbol{G}(11) = \frac{1}{2}\begin{bmatrix} D^2 & D^2 \\ D^2 & D^2 \end{bmatrix}$$

Using (7.20) we obtain the matrix transfer function of the Error State Diagram as

$$\boldsymbol{G} = \frac{1}{2}\frac{D^6}{1-D^6}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{7.21}$$

The scalar transfer function, $T(D)$, is then given by

$$T(D) = \frac{1}{2} 1^T \boldsymbol{G} 1 = \frac{D^6}{1-D^2} \tag{7.22}$$

The upper bound on the probability of error can be computed by substituting $D = e^{-\left\{\frac{1}{4N_0}\right\}}$ in (7.22).

$$P_e \leq \frac{D^6}{1-D^2}\bigg|_{D = e^{\frac{-1}{4N_0}}} \tag{7.23}$$

Example 7.11 Consider another rate 1/2 TCM scheme with $m = 1$, and $M = 4$. The two state Trellis Diagram is same as the one in the previous example. However, the symbol allocation from the 4-PSK Constellation is different, and is given in Fig. 7.17.

Fig. 7.17

Note that this symbol assignment violates the Ungerboek design principles. Let us again denote the error vector by $e = (e_2\ e_1)$. The Error State Diagram for this TCM scheme is given in Fig. 7.18.

Fig. 7.18

The matrix transfer function of the Error State Diagram is

$$G = G(11)[I_2 - G(10)]^{-1}\, G(01) \tag{7.24}$$

where I_2 is the 2×2 identity matrix. In this case there are only three error vectors possible, {01, 10,11}. From (7.19) we calculate

$$G(01) = \frac{1}{2}\begin{bmatrix} D^2 & D^2 \\ D^2 & D^2 \end{bmatrix},\ G(10) = \frac{1}{2}\begin{bmatrix} D^4 & D^4 \\ D^4 & D^4 \end{bmatrix},\ \text{and } G(11) = \frac{1}{2}\begin{bmatrix} D^2 & D^2 \\ D^2 & D^2 \end{bmatrix}$$

Using (7.23) we obtain the matrix transfer function of the Error State Diagram as

$$G = \frac{1}{2}\frac{D^4}{1-D^4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{7.25}$$

The scalar transfer function, $T(D)$, is then given by

$$T(D) = \frac{1}{2} 1^T G 1 = \frac{D^4}{1-D^4} \tag{7.26}$$

The upper bound on the probability of error is

$$P_e = \left.\frac{D^4}{1-D^4}\right|_{D = e^{\frac{-1}{4N_0}}} \tag{7.27}$$

Comparing (7.23) and (7.27) we observe that, simply by changing the symbol assignment to the branches of theTrellis Diagram, we degrade the performance considerably. In the second example, the upper bound on the error probability has loosened by two orders of magnitude *(assuming $D \ll 1$*, i.e., for the high SNR case).

A tighter upper bound on the error event probability is given by (exercise)

$$P_e \le \frac{1}{2} efrc\left(\sqrt{\frac{d^2_{free}}{4N_0}}\right) e^{\frac{d^2_{free}}{4N_0}}\ T(D)|_{D = e^{\frac{-1}{4N_0}}}. \tag{7.28}$$

From (7.28), an asymptotic estimate on the error event probability can be obtained by considering only the error events with free Euclidean Distance

$$P_e \approx \frac{1}{2} N(d_{free})\ efrc\left(\sqrt{\frac{D^2_{free}}{4N_0}}\right) \tag{7.29}$$

The bit error probability can be upper bounded simply by weighting the pairwise error probabilities by the number of incorrect input bits associated with each error vector and then dividing the result by m. Therefore,

$$P_b \le \frac{1}{m}\frac{\partial T(D,I)}{\partial I}|_{I=1,\, D = e^{\frac{-1}{4N_0}}}, \tag{7.30}$$

Where $T(D, I)$ is the **augmented generating** function of the modified State Diagram. The concept of the Modified State Diagram was introduced in the chapter on Convolutional Codes (Section 6.5). A tighter upper bound can also be obtained for the bit error probability, and is given by

$$P_e \le \frac{1}{2m} efrc\left(\sqrt{\frac{d^2_{free}}{4N_0}}\right) e^{\frac{D^2_{free}}{4N_0}}\ \left.\frac{\partial T(D,I)}{\partial I}\right|_{I=1,\, D=e^{\frac{1}{4N_0}}} \tag{7.31}$$

From (7.31), we observe that the upper bound on the bit error probability strongly depends on d_{free}. In the next section, we will learn some methods for estimating d_{free}.

7.7 COMPUTATION OF d_{free}

We have seen that the *Euclidean* Free Distance, d_{free}, is the singlemost important parameter for determining how good a TCM scheme is for AWGN channels. It defines the asymptotic coding gain of the scheme. In chapter 6 (Section 6.5) we saw that the generating function can be used to calculate the *Hamming* Free Distance d_{free}. The transfer function of the error state diagram, $T(D)$, includes information about the distance of all the paths in the trellis from the all zero path. If $T(D)$ is obtained in a closed form, the value of d_{free} follows immediately from the expansion of the function in a power series. The generating function can be written as

$$T(D) = N(d_{free}) D^{d^2_{free}} + N(d_{next}) D^{d^2_{next}} + \ldots \tag{7.32}$$

where d^2_{next} is the second smallest squared Euclidean Distance. Hence the smallest exponent of D in the series expansion is d_{free}. However, in most cases, a closed form expression for $T(D)$ may not be available, and one has to resort to numerical techniques.

Consider the function

$$\phi_1(D) = \ln\left[\frac{T(\varepsilon D)}{T(D)}\right] \tag{7.33}$$

$\phi_1(D)$ decreases monotonically to the limit d^2_{free} as $D \to 0$. Therefore we have an upper bound on d^2_{free} provided $D > 0$. In order to obtain a lower bound on d^2_{free} consider the following function

$$\phi_2(D) = \frac{\ln T(D)}{\ln D} \tag{7.34}$$

Taking logarithm on both sides of (7.32) we get,

$$d^2_{free} \ln D = \ln T(D) - \ln N(d_{free}) - \ln\left[1 + \frac{N(d_{free})}{N(d_{next})} D^{d^2_{next} - d^2_{free}} \ldots\right] \tag{7.35}$$

If we take $D \to 0$, provided $D > 0$, from (7.34) and (7.35) we obtain

$$\frac{\ln T(D)}{\ln D} = d^2_{free} - \varepsilon(D) \tag{7.36}$$

where, $\varepsilon(D)$ is a function that is greater than zero, and tends to zero monotonically as $D \to 0$. Thus, if we take smaller and smaller values of $\phi_1(D)$ and $\phi_2(D)$, we can obtain values that are extremely close to d^2_{free}

It should be kept in mind that even though d^2_{free} is the single most important parameter to determine the quality of a TCM scheme, two other parameters are also influential:

(i) The **error coefficient** $N(d_{free})$: A factor of two increase in this error coefficient reduces the coding gain by approximately 0.2 dB for error rates of 10^{-6}.

(ii) The **next distance** d_{next} : is the second smallest Euclidean Distance between two paths forming an error event. If d_{next} is very close to d_{free}, the SNR requirement for good approximation of the upper bound on P_e may be very large.

So far, we have focussed primarily on AWGN channels. We found that the best design strategy is to maximize the free Euclidean Distance, d_{free}, for the code. In the next section, we consider the design rules for TCM over fading channels. Just to remind the readers, fading channels are frequently encountered in radio and mobile communications. One common cause of fading is the multipath nature of the propagation medium. In this case, the signal arrives at the receiver from different paths (with time varying nature) and gets added together. Depending on whether the signals from different paths add up in phase, or out of phase, the net received signal exhibits a random variation in amplitude and phase. The drops in the received signal amplitude (below a threshold) are called **fades**.

7.8 TCM FOR FADING CHANNELS

In this section we will consider the performance of trellis coded M-ary Phase Shift Keying (MPSK) over a Fading Channel. We know that a TCM encoder takes in an input bit stream and

outputs a sequence of symbols. In this treatment we will assume that each of these symbols s_i belong to the MPSK signal set. By using complex notation, each symbol can be represented by a point in the complex plane. The coded signals are **interleaved** in order to spread the burst of errors caused by the slowly varying fading process. These interleaved symbols are then pulse-shaped for no **inter-symbol interference** and finally translated to RF frequencies for transmission over the channel. The channel corrupts these transmitted symbols by adding a fading gain (which is a negative gain, or a positive loss, depending on one's outlook) and AWGN. At the receiver end, the received sequences are demodulated and quantized for soft decision decoding. In many implementations, the channel estimator provides an estimate of the channel gain, which is also termed as the **channel state information**. Thus we can represent the received signal at time i as

$$r_i = g_i \times s_i + n_i, \tag{7.37}$$

where n_i is a sample of the zero mean Gaussian noise process with variance $N_0/2$ and g_i is the complex channel gain, which is also a sample of a complex Gaussian process with variance σ^2_g. The complex channel gain can be explicitly written using the phasor notation as follows

$$g_i = a_i\, e^{j\phi_i}, \tag{7.38}$$

where a_i and ϕ_i are the amplitude and phase processes respectively. We now make the following assumptions:

(i) The receiver performs coherent detection,

(ii) The interleaving is ideal, which implies that the fading amplitudes are statistically independent and the channel can be treated as memoryless.

Thus, we can write

$$r_i = a_i \cdot s_i + n_i \tag{7.39}$$

We know that for a channel with a diffused multipath and no direct path the fading amplitude is Rayleigh distributed with **Probability Density Function (pdf)**

$$p_A(a) = 2ae^{-a^2},\ a \geq 0 \tag{7.40}$$

For the case when there exists a direct path in addition to the multipath, **Rician Fading** is observed. The pdf of the Rician Fading Amplitude is given by

$$p_A(a) = 2a(1+K)e^{-(K+a^2(k+1)}I_0\left(2a\sqrt{K(1+K)}\right), \tag{7.41}$$

where $I_0(.)$ is the zero-order, modified Bessel Function of the first kind and K is the Rician Parameter defined as follows.

Definition 7.8 The **Rician Parameter** K is defined as the ratio of the energy of the direct component to the energy of the diffused multipath component. For the extreme case of $K = 0$, the pdf of the Rician distribution becomes the same as the pdf of the Rayleigh Distribution.

We now look at the performance of the TCM scheme over a fading channel. Let $\mathbf{r}_l = (r_1, r_2, \ldots, r_{el})$ be the received signal. The maximum likely decoder, which is usually implemented by the Viterbi decoder, chooses the coded sequence that most likely corresponds to the received signals. This is achieved by computing a metric between the sequence of received signals, $\mathbf{r}_e$ and the possible transmitted signals, $\mathbf{s}_t$ As we have seen earlier, this metric is related to the conditional channel probabilities

$$m(\mathbf{r}_l, \mathbf{s}_l) = \ln p(\mathbf{r}_l | \mathbf{s}_l) \tag{7.42}$$

If the channel state information is being used, the metric becomes

$$m(\mathbf{r}_l, \mathbf{s}_l; \hat{\mathbf{a}}_l) = \ln p(\mathbf{r}_l | \mathbf{s}_l, \hat{\mathbf{a}}_l) \tag{7.43}$$

Under the assumption of ideal interleaving, the channel is memoryless and hence the metrics can be expressed as the following summations

$$m(\mathbf{r}_l, \mathbf{s}_l) = \sum_{i=1}^{l} \ln p(r_l | s_l) \tag{7.44}$$

and

$$m(r_l, b_l; \hat{a}_l) = \sum_{i=1}^{l} \ln p(r_l | s_l, \hat{a}_l) \tag{7.45}$$

First, we consider the scenario where the channel state information is known, i.e., $\hat{a}_i = a_i$. The metric can be written as

$$m(r_i, s_i; \hat{a}_i) = -|r_i - a_i s_i|^2 \tag{7.46}$$

Therefore, the pairwise error probability is given by

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l) = E_{al}[P_2(\mathbf{s}_b \hat{\mathbf{s}}_l | \mathbf{a}_l)], \tag{7.47}$$

where

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l | \mathbf{a}_l) = P[m(\mathbf{r}_b \hat{\mathbf{s}}_l; \mathbf{a}_l) \geq m(\mathbf{r}_b \mathbf{s}_l; \mathbf{a}_l) | \mathbf{a}_l)], \tag{7.48}$$

and E is the statistical expectation operator. Using the Chernoff Bound, the pairwise error probability can be upper bounded as follows.

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l) \leq \prod_{i=1}^{l} \frac{1+K}{1+K+\frac{1}{4N_0}|s_i - \hat{s}_i|} \exp\left[-\frac{K\frac{1}{4N_0}|s_i - \hat{s}_i|^2}{1+K\frac{1}{4N_0}|s_i - \hat{s}_i|^2}\right] \tag{7.49}$$

For high SNR, the above equation simplifies to

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l) \leq \prod_{i \in \eta} \frac{(1+K)e^{-K}}{\frac{1}{4N_0}|s_i - \hat{s}_i|^2} \tag{7.50}$$

where $\boldsymbol{\eta}$ is the set of all i for which $\mathbf{s}_i \neq \hat{\mathbf{s}}_i$. Let us denote the number of elements in $\boldsymbol{\eta}$ by l_η, then we can write

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l) \leq \frac{\left((1+K)e^{-K}\right)^{l_\eta}}{\left(\frac{1}{4N_0}\right)^{l_\eta} d_p^2(l_\eta)} \tag{7.51}$$

where

$$d_p^2(l_\eta) \leq \prod_{i \in \eta} |s_i - \hat{s}_i|^2 \tag{7.52}$$

is the **squared product distance** of the signals $\mathbf{s}_i \neq \hat{\mathbf{s}}_i$. The term l_η is called the **effective length** of the error event $(\mathbf{s}_l, \neq \hat{\mathbf{s}}_l)$. A union bound on the error event probability P_e has already been discussed before. For the high SNR case, the upper bound on P_e can be expressed as

$$P_e \leq \sum_{l_\eta} \sum_{d_p^2(l_\eta)} \alpha[l_\eta, d_p^2(l_\eta)] \frac{\left((1+K)e^{-K}\right)^{l_\eta}}{\left(\frac{1}{4N_0}\right)^{l_\eta} d_p^2(l_\eta)} \tag{7.53}$$

where $\alpha[l_\eta, d_p^2(l_\eta)]$ is the average number of code sequences having the effective length l_η and the squared product distance $d_p^2(l_\eta)$. The error event probability is actually dominated by the smallest effective length l_η and the smallest product distance $d_p^2(l_\eta)$. Let us denote the smallest effective length l_η by L and the corresponding product distance by $d_p^2(l_\eta)$. The error event probability can then be asymptotically approximated by

$$P_e \approx \alpha(L, d_p^2(L)) \frac{\left((1+K)e^{-K}\right)^{L}}{\left(\frac{1}{4N_0}\right)^{L} d_p^2(L)}. \tag{7.54}$$

We make the following observations from (7.54)

(i) The error event probability asymptotically varies with the Lth power of SNR. This is similar to what is achieved with a time diversity technique. Hence, L is also called the time diversity of the TCM scheme.

(ii) The important TCM design parameters for fading Channel are the time diversity, L, and the product distance $d_p^2(L)$. This is in contrast to the free Euclidean Distance parameter for AWGN channel.

(iii) TCM codes designed for AWGN channels would normally fare poorly in fading channels and vice versa.

(iv) For large values of the Rician parameter, K, the effect of the free Euclidean Distance on the performance of the TCM scheme becomes dominant.

(v) At low SNR, again, the free Euclidean Distance becomes important for the performance of the TCM scheme.

Thus the basic *design rules* for TCMs for fading channels, at high SNR and for small values of K, are

(i) maximize the effective length, L, of the code, and
(ii) minimize the minimum product distance $d_p^2(L)$.

Consider a TCM scheme with effective length, L, and the minimum product distance $d_{p_1}^2(L)$. Suppose the code is redesigned to yield a minimum product distance, $d_{p_2}^2(L)$ with the same L. The increase in the coding gain due the increase in the minimum product distance is given by

$$\Delta_g = SNR_1 - SNR_2 \,|_{P_{e_1} - P_{e_2}} = \frac{10}{L} \log \frac{d_{p_2}^2(L)\alpha_1}{d_{p_1}^2(L)\alpha_2}, \tag{7.55}$$

where a_i, $i = 1, 2$, is the average number of code sequences with effective length L for the TCM scheme i. We observe that for a fixed value of L, increasing the minimum product distance corresponding to a smaller value of L is more effective in improving the performance of the code.

So far, we have assumed that the channel state information was available. A similar analysis as carried out for the case where channel state information was available can also be done when the information about the channel is unavailable. In the *absence* of channel state information, the metric can be expressed as

$$m(r_i, s_i; \hat{a}_i) = -|r_i - s_i|^2. \tag{7.56}$$

After some mathematical manipulations, it is shown that

$$P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l) \le \frac{(2e/l_\eta)^{l_\eta}}{(1/N_0)^{l_\eta}} \frac{\left(\sum_{i \in l_\eta} |s_i - \hat{s}_i|^2\right)^{l_\eta}}{d_p^4(l_\eta)} (1+K)^{l_\eta} e^{-l_\eta K} \tag{7.57}$$

Using arguments discussed earlier in this section, the error event probability P_e can be determined for this case when the channel state information is not available.

7.9 CONCLUDING REMARKS

Coding and modulation were first analyzed together as a single entity by Massey in 1974. Prior to that time, in all coded digital communications systems, the encoder/decoder and the modulator/demodulator were designed and optimized separately. Massey's idea of combined coding and modulation was concretized in the seminal paper by Ungerboeck in 1982. Similar ideas were also proposed earlier by Imai and Hirakawa in 1977, but did not get due attention. The primary advantage of TCM was its ability to achieve increased power efficiency without the customary increase in the bandwidth introduced by the coding process. In the following years the theory of TCM was formalized by different researchers. Calderbank and Mazo showed that the asymmetric one-dimensional TCM schemes provide more coding gain than symmetric TCM schemes. Rotationally invariant TCM schemes were proposed by Wei in 1984, which were subsequently adopted by CCITT for use in the new high speed voiceband modems.

SUMMARY

- The Trellis Coded Modulation (TCM) Technique allows us to achieve a better performance *without* bandwidth expansion or using extra power.
- The minimum Euclidean Distance between any two paths in the trellis is called the free Euclidean Distance, d_{free} of the TCM scheme.
- The difference between the values of the *SNR* for the coded and uncoded schemes required to achieve the same error probability is known as the coding gain, $g = \text{SNR}|_{uncoded} - \text{SNR}|_{coded}$. At high *SNR*, the coding gain can be expressed as $g_\infty = g|_{SNR \to \infty} = 10 \log \frac{\left(d_{free}^2/E_s\right)_{coded}}{\left(d_{free}^2/E_s\right)_{uncoded}}$, where g_∞ represents the Asymptotic Coding Gain and E_s is the average signal energy.
- The mapping by Set Partitioning is based on successive partitioning of the expanded 2^{m+1}-ary signal set into subsets with increasing minimum Euclidean Distance. Each time we partition the set, we reduce the number of the signal points in the subset, but increase the minimum distance between the signal points in the subset.
- Ungerboeck's TCM design rules (based on heuristics) for AWGN channels are

  Rule 1: Parallel transitions, if present, must be associated with the signals of the subsets in the lowest layer of the Set Partitioning Tree. These signals have the minimum Euclidean Distance $\Delta_{\tilde{m}+1}$.

  Rule 2: The transitions originating from or merging into one state must be associates with signals of the first step of set partitioning. The Euclidean distance between these signals is at least Δ_1.

  Rule 3: All signals are used with equal frequency in the Trellis Diagram.
- The Viterbi Algorithm can be used to decode the received symbols for a TCM scheme. The branch metric used in the decoding algorithm is the *Euclidean Distance* between the received signal and the signal associated with the corresponding branch in the trellis.
- The average number of nearest neighbours at free distance, $N(d_{free})$, gives the average number of paths in the trellis with free Euclidean Distance d_{free} from a transmitted sequence. This number is used in conjunction with d_{free} for the evaluation of the error event probability.
- The probability of error $P_e \le T(D)|_{D=e^{-1/4N_0}}$, where, $T(D) = \frac{1}{N}\mathbf{1}^T\boldsymbol{G}\mathbf{1}$, and the matrix $\boldsymbol{G} = \sum_{l=1}^{\infty} \sum_{\boldsymbol{E}_l \neq 0} \prod_{n=1}^{l} G(e_n)$. $T(D)$ is the scalar transfer function. A tighter upper bound on the error event probability is given by $P_e \le \frac{1}{2} erfc\left(\sqrt{\frac{d_{free}^2}{4N_0}}\right) e^{\frac{d_{free}^2}{4N_0}} T(D)\Big|_{D=e^{\frac{-1}{4N_0}}}$

- For fading channels, $P_2, (s_i, \hat{s}) \le \dfrac{((1+K)e^{-K})^{l_\eta}}{\left(\dfrac{1}{4N_0}\right)^{l_\eta} d_p^2(l_\eta)}$ where $d_p^2(l_\eta) \le \prod_{i \in \eta} |s_i - \hat{s}_i|^2$. The term l_η is the effective length of the error event $(s_l, \hat{s}_l)$ and K is the Rician parameter. Thus, the error event probability is dominated by the smallest effective length l_η and the smallest product distance $d_p^2(l_\eta)$.
- The design rules for TCMs for fading channels, at high SNR and for small values of K, are
  (i) maximize the effective length, L, of the code, and
  (ii) minimize the minimum product distance $d_p^2(L)$.
- The increase in the coding gain due to increase in minimum product distance is given by

$$\Delta_g = SNR_1 - SNR_2|_{P_{e1} - P_{e2}} = \frac{10}{L} \log \frac{d_{p2}^2(L)\alpha_1}{d_{p1}^2(L)\alpha_2},$$ where a_i, $i = 1, 2$, is the average number of code sequences with effective length L for the TCM scheme i.

> *A little inaccuracy sometimes saves a ton of explanation*
> *H. H. Munro (Saki) (1870-1916)*

PROBLEMS

7.1 Consider a rate 2/3 Convolutional Code defined by

$$G(D) = \begin{bmatrix} 1 & D & D+D^2 \\ D^2 & 1+D & 1+D+D^2 \end{bmatrix}$$

This code is used with an 8PSK signal set that uses Gray Coding (the three bits per symbol are assigned such that the codes for two adjacent symbols differ only in 1 bit location). The throughput of this TCM scheme is 2 bits/sec/Hz.

(a) How many states are there in the Trellis Diagram for this encoder?
(b) Find the free Euclidean Distance.
(c) Find the Asymptotic coding gain with respect to uncoded QPSK, which has a throughput of 2 bits/sec/Hz.

7.2 In Problem 7.1, suppose instead of Gray Coding, natural mapping is performed, i.e., $s_0 \to 000$, $s_1 \to 001$, ..., $s_7 \to 111$.

(a) Find the free Euclidean Distance.
(b) Find the Asymptotic coding gain with respect to uncoded QPSK (2 bits/sec/Hz).

7.3 Consider the TCM encoder shown in Fig. 7.19.

Fig. 7.19 *Figure for Problem 7.3.*

(a) Draw the State Diagram for this encoder.
(b) Draw the Trellis Diagram for this encoder.
(c) Find the free Euclidean Distance, d_{free}^2. In the Trellis Diagram, show one pair of two paths which result in d_{free}^2. What is $N(d_{free}^2)$?
(d) Next, use set partitioning to assign the symbols of 8-PSK to the branches of the Trellis Diagram. What is the d_{free}^2 now?
(e) Encode the following bit stream using this encoder: 1 0 0 1 0 0 0 1 0 1 0 ... Give your answer for both the natural mapping and mapping using Set Partitioning.
(f) Compare the asymptotic coding gains for the two different kinds of mapping.

7.4 We want to design a TCM scheme that has a 2/3 convolutional encoder followed by a signal mapper. The mapping is done based on set partitioning of the Asymmetric Constellation Diagram shown below. The trellis is a four-state, fully connected trellis.

(a) Perform Set Partitioning for the following Asymmetric Constellation Diagram.
(b) What is the free Euclidean distance, d_{free}^2, for this asymmetric TCM scheme? Compare it with the d_{free}^2 for the case when we use the standard 8-PSK Signal Constellation.
(c) How will you choose the value of θ for improving the performance of the TCM scheme using the Asymmetric Signal Constellation shown in Fig. 7.20?

Fig. 7.20 *Figure for Problem 7.4.*

7.5 Consider the rate ¾ encoder shown in Fig. 7.21. The four output bits from the encoder are mapped onto one of the sixteen possible symbols from the Constellation Diagram shown below. Use Ungerboeck's design rules to design a TCM scheme for an AWGN channel. What is the asymptotic coding gain with respect to uncoded 8-PSK?

Fig. 7.21 *Figure for Problem 7.5.*

7.6 Consider the expression for pairwise error probability over a Rician Fading Channel.

$$P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l) \le \prod_{i=1}^{l} \frac{1+K}{1+K+\frac{1}{4N_0}|s_i-\hat{s}_i|^2} \exp\left[-\frac{K\frac{1}{4N_0}|s_i-\hat{s}_i|^2}{1+K+\frac{1}{4N_0}|s_i-\hat{s}_i|^2}\right]$$

(a) Show that the above inequality can be simplified for a large value of K to

$$P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l) \le \prod_{i=1}^{l} \exp\left[-\frac{1}{4N_0}|s_i-\hat{s}_i|^2\right]$$

Comment.

(b) Show that for low SNR the original inequality may be expressed as

$$P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l) \le \exp\left[\frac{d_E^2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l)}{4N_0}\right]$$

7.7 Consider a TCM scheme designed for a Rician Fading Channel with an effective length L and the minimum product distance $d_p^2(L)$. Suppose, we wish to redesign this code to obtain an improvement of 3 dB in SNR.

(a) Compute the desired effective length L if the $d_p^2(L)$ is kept unchanged.

(b) Compute the desired product distance $d_p^2(L)$ if the effective length L is kept unchanged.

7.8 Suppose you have to design a TCM scheme for an AWGN channel (SNR = γ). The desired BER is P_e. Draw a flowchart as to how you will go about designing such a scheme.

(a) How many states will there be in your Trellis?

(b) How will you design the convolutional encoder?

(c) Would you have parallel paths in your design?

(d) What kind of modulation scheme will you choose and why?

(e) How will you assign the symbols of the modulation scheme to the branches?

7.9 For Viterbi decoding the metric used is of the form

$$m(\boldsymbol{r}_l, \boldsymbol{s}_l) = \ln p(\boldsymbol{r}_l|\boldsymbol{s}_l).$$

(a) What is the logic behind choosing such a metric?

(b) Suggest another metric that will be suitable for fading channels. Give reasons for your answer.

7.10 A TCM scheme designed for a Rician Fading Channel ($K = 3$) and a high SNR environment (SNR = 20 dB) has $L = 5$ and $d_p^2(L) = 2.34\, E_s^2$. It has to be redesigned to produce an improvement of 2 dB.

(a) What is the $d_p^2(L)$ of the new code?

(b) Comment on the new d_{free}^2.

7.11 Consider the TCM scheme shown in Fig. 7.22 consisting of a rate ½ convolutional encoder coupled with a mapper.

(a) Draw the Trellis Diagram for this encoder.

(b) Determine the scalar transfer function, $T(D)$.

(c) Determine the augmented generating function, $T(D, L, I)$.

(d) What is the minimum Hamming Distance (d_{free}^H) of this code?

(e) How many paths are there with this d_{free}^H?

Fig. 7.22 *Figure for Problem 7.11.*

7.12 Consider the pairwise error probability $P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l)$.

(a) For a maximum likelihood decoder, prove that

$$P_2(\boldsymbol{s}_l, \hat{\boldsymbol{s}}_l) = \int_{\boldsymbol{r}} f(\boldsymbol{r}) p_{R|S}(\boldsymbol{r}|\boldsymbol{s}_l)\, d\boldsymbol{r}$$

where $\boldsymbol{r}$ is the received vector, $p_{R|S}(\boldsymbol{r}|\boldsymbol{s}_l)$ is the channel transition probability density function and

$$f(\boldsymbol{r}) = \begin{cases} 1 & \text{if } p_{R|S}(\boldsymbol{r}|\hat{\boldsymbol{s}}_l) \ge p_{R|S}(\boldsymbol{r}|\boldsymbol{s}_l) \\ 0 & \text{otherwise} \end{cases}$$

(b) Show that

$$f(\mathbf{r}) \le \frac{p_{R|S}(\mathbf{r}|\hat{\mathbf{s}}_l)}{p_{R|S}(\mathbf{r}|\hat{\mathbf{s}}_l)}$$

$$P_2(\mathbf{s}_l, \hat{\mathbf{s}}_l) \le \int_{\mathbf{r}} \sqrt{p_{R|S}(\mathbf{r}|\hat{\mathbf{s}}_l)\, p_{R|S}(\mathbf{r}|\mathbf{s}_l)}\, d\mathbf{r}$$

COMPUTER PROBLEMS

7.13 Write a computer program to perform trellis coded modulation, given the trellis structure and the mapping rule. The program should take in an input bit stream and output a sequence of symbols. The input to the program may be taken as two matrices, one that gives the connectivity between the states of the trellis (essentially the structure of the trellis) and the second, which gives the branch labels.

7.14 Write a computer program to calculate the squared free Euclidean distance d^2_{free}, the effective length L, and the minimum product distance, $d^2_p(L)$, of a TCM Scheme, given the Trellis Diagram and the labels on the branches.

7.15 Write a computer program that performs Viterbi decoding on an input stream of symbols. This program makes use of a given trellis and the labels on the branches of the Trellis Diagram.

7.16 Verify the performance of the different TCM schemes given in this chapter in AWGN environment. To do so, take a long chain of random bits and input it to the TCM encoder. The encoder will produce a sequence of symbols (analog waveforms). Corrupt these symbols with AWGN of different noise power, i.e., simulate scenarios with different SNRs. Use Viterbi decoding to decode the received sequence of corrupted symbols (distorted waveforms). Generate a plot of the BER versus the SNR and compare it with the theoretically predicted error rates.

7.17 Write a program to observe the effect of decoding window size for the Viterbi decoder. Generate a plot of the error rate versus the window size. Also plot the number of computations versus the window size.

7.18 Write a computer program that performs exhaustive search in order to determine a rate 2/3 TCM encoder which is designed for AWGN (maximize d^2_{free}). Assume that there are four states in the Trellis Diagram and it is a fully connected trellis. The branches of this trellis are labelled using the symbols from an 8-PSK signal set. Modify the program to perform exhaustive search for a good TCM scheme with a four-state trellis with the possibility of parallel branches.

7.19 Write a computer program that performs exhaustive search in order to determine a rate 2/3 TCM encoder which is designed for a fading channel (maximize $d_p^2(L)$). Assume that there are four states in the trellis diagram and it is a fully connected trellis. The branches of this trellis are labelled using the symbols from an 8-PSK signal set. List out the $d_p^2(L)$ and L of some of the better codes found during the search.

7.20 Draw the family of curves depicting the relation between P_e and L_{eff} for different values of K (Rician Parameter) for

(a) High SNR,

(b) Low SNR.

Comment on the plots.

8

Cryptography

> *One cannot escape the feeling that these mathematical formulae have an independent existence and an intelligence of their own, that they are wiser than we are, wiser even than their discoverers, that we get more out of them than we originally put in to them.*
>
> *Heinrich Hertz 1857-1894*

8.1 INTRODUCTION TO CRYPTOGRAPHY

Cryptography is the science of devising methods that allow information to be sent in a secure form in such a way that the only person able to retrieve this information is the intended recipient. Encryption is based on algorithms that scramble information into unreadable or non-discernible form. Decryption is the process of restoring the scrambled information to its original form (see Fig. 8.1).

A **Cryptosystem** is a collection of algorithms and associated procedures for hiding and revealing (un-hiding!) information. **Cryptanalysis** is the process (actually, the art) of analyzing a cryptosystem, either to verify its integrity or to break it for ulterior motives. An **attacker** is a person or system that performs unauthorised cryptanalysis in order to break a cryptosystem. Attackers are also referred to as hackers, interlopers or eavesdroppers. The process of attacking a cryptosystem is often called **cracking**.

The job of the **cryptanalyst** is to find the weaknesses in the cryptosystem. In many cases, the developers of a cryptosystem announce a public challenge with a large prize-money for anyone

who can crack the scheme. Once a cryptosystem is broken (and the cryptanalyst discloses his techniques), the designers of the scheme try to strengthen the algorithm. Just because a cryptosystem has been broken does not render it useless. The hackers may have broken the system under optimal conditions using equipment (fast computers, dedicated microprocessors, etc.) that is usually not available to common people. Some cryptosystems are rated in terms of the length of time and the price of the computing equipment it would take to break them!

In the last few decades, cryptographic algorithms, being mathematical in nature, have become so advanced that they can only be handled by computers. This, in effect, means that the uncoded message (prior to encryption) is binary in form, and can therefore be anything; a picture, a voice, a text such as an e-mail or even a video.

Fig. 8.1 *The Process of Encryption and Decryption.*

Cryptography is not merely used for military and diplomatic communications as many people tend to believe. In reality, cryptography has many commercial uses and applications. From protecting confidential company information, to protecting a telephone call, to allowing someone to order a product on the Internet without the fear of their credit card number being intercepted and misused, cryptography is all about increasing the level of privacy of individuals and groups. For example, cryptography is often used to prevent forgers from counterfeiting winning lottery tickets. Each lottery ticket can have two numbers printed onto it, one plaintext and one the corresponding cipher. Unless the counterfeiter has cryptanalyzed the lottery's cryptosystem he or she will not be able to print an acceptable forgery.

The chapter is organized as follows. We begin with an overview of different encryption techniques. We will, then, study the concept of secret-key cryptography. Some specific secret-key cryptographic techniques will be discussed in detail. The public-key cryptography will be introduced next. Two popular public-key cryptographic techniques, the RSA algorithm and PGP, will be discussed in detail. A flavour of some other cryptographic techniques in use today will also be given. The chapter will conclude with a discussion on cryptanalysis and the politics of cryptography.

8.2 AN OVERVIEW OF ENCRYPTION TECHNIQUES

The goal of a cryptographic system is to provide a high level of confidentiality, integrity, non-repudiability and authenticity to information that is exchanged over networks.

Confidentiality of messages and stored data is protected by hiding information using encryption techniques. Message integrity ensures that a message remains unchanged from the time it is created to the time it is opened by the recipient. Non-repudiation can provide a way of proving that the message came from someone even if they try to deny it. Authentication provides two services. First, it establishes beyond doubt the origin of the message. Second, it verifies the identity of a user logging into a system and continues to verify their identity in case someone tries to break into the system.

Definition 8.1 A message being sent is known as **plaintext**. The message is coded using a Cryptographic Algorithm. This process is called **encryption**. An encrypted message is known as **ciphertext**, and is turned back into plaintext by the process of **decryption**.

It must be assumed that any eavesdropper has access to all communication between the sender and the recipient. A method of encryption is only secure if even with this complete access, the eavesdropper is still unable to recover the original plaintext from the ciphertext.

There is a big difference between *security* and *obscurity*. Suppose, a message is left for somebody in an airport locker, and the details of the airport and the locker number are known only to the intended recipient, then this message is not secure, merely obscure. If however, all potential eavesdroppers know the exact location of the locker, and they still cannot open the locker and access the message, then this message is secure.

Definition 8.2 A **key** is a value that causes a Cryptographic Algorithm to run in a specific manner and produce a specific ciphertext as an output. The key size is usually measured in bits. The bigger the key size, the more secure will be the algorithm.

Example 8.1 Suppose we have to encrypt and send the following stream of binary data (which might be originating from voice, video, text or any other source)

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

We can use a 4-bit long key, $x = 1011$, to encrypt this bit stream. To perform encryption, the plaintext (binary bit stream) is first subdivided in to blocks of 4 bits.

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

Each sub-block is XORed (binary addition) with the key, $x = 1011$. The encrypted message will be

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

The recipient must also possess the knowledge of the key in order to decrypt the message. The decryption process is fairly simple in this case. The ciphertext (the received binary bit stream) is first subdivided into blocks of 4 bits. Each sub-block is XORed with the key, $x = 1011$. The decrypted message will be the original plaintext

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

It should be noted that just one key is used both for encryption and decryption.

Example 8.2 Let us devise an algorithm for text messages, which we shall call *character* + x. Let $x = 5$. In this encryption technique, we replace every alphabet by the fifth one following it, i.e., A becomes F, B becomes G, C becomes H, and so on. The recipients of the encrypted message just need to know the value of the key, x, in order to decipher the message. The key must be kept separate from the encrypted message being sent. Because there is just one key which is used for encryption and decryption, this kind of technique is called **Symmetric Cryptography** or **Single Key Cryptography** or **Secret Key Cryptography**. The problem with this technique is that the key has to be kept confidential. Also, the key must be changed from time to time to ensure secrecy of transmission. This means that the secret key (or the set of keys) has to be communicated to the recipient. This might be done physically.

To get around this problem of communicating the key, the concept of **Public Key Cryptography** was developed by Difie and Hellman. This technique is also called the **Asymmetric Encryption**. The concept is simple. There are two keys, one is held privately and the other one is made public. What one key can lock, the other key can unlock.

Example 8.3 Suppose we want to send an encrypted message to recipient A using the public key encryption technique. To do so we will use the public key of the recipient A and use it to encrypt the message. When the message is received, recipient A decrypts it with his private key. Only the private key of recipient A can decrypt a message that has been encrypted with his public key. Similarly, recipient B can only decrypt a message that has been encrypted with his public key. Thus, no private key ever needs to be communicated and hence one does not have to trust any communication channel to convey the keys.

Let us consider another scenario. Suppose we want to send somebody a message and also provide a proof that the message is actually from us (a lot of harm can be done by providing bogus information, or rather, misinformation!). In order to keep a message private and also provide authentication (that it is indeed from us), we can perform a special encryption on the plain text with our *private* key, then encrypt it again with the *public* key of the recipient. The recipient uses his private key to open the message and then use our public key to verify the authenticity. This technique is said to use **Digital Signatures**.

There is another important encryption technique called the **One-way Function**. It is a non-reversible quick encryption method. The encryption is easy and fast, but the decryption is not. Suppose we send a document to recipient A and want to check at a later time whether the document has been tampered with. We can do so by running a one-way function, which produces a fixed length value called a **hash** (also called the **message digest**). The hash is the unique signature of the document that can be sent along with the document. Recipient A can run the same one-way function to check whether the document has been altered.

The actual mathematical function used to encrypt and decrypt messages is called a **Cryptographic Algorithm or cipher**. This is only a part of the system used to send and receive secure messages. This will become clearer as we discuss specific systems in detail.

As with most historical ciphers, the security of the message being sent relies on the algorithm itself remaining secret. This technique is known as a **Restricted Algorithm**. It has the following fundamental drawbacks.

(i) The algorithm obviously has to be restricted to only those people that you want to be able to decode your message. Therefore a new algorithm must be invented for every discrete group of users.

(ii) A large or changing group of users cannot utilise them, as every time one user leaves the group, everyone must change the algorithm.

(iii) If the algorithm is compromised in any way, a new algorithm must be implemented.

Because of these drawbacks, Restricted Algorithms are no longer popular and have given way to key-based algorithms.

Practically all modern cryptographic systems make use of a key. Algorithms that use a key allow all details of the algorithm to be widely available. This is because all of the security lies in the key. With a key-based algorithm the plaintext is encrypted and decrypted by the algorithm which uses a certain key, and the resulting ciphertext is dependent on the key, and not the algorithm. This means that an eavesdropper can have a complete copy of the algorithm in use, but without the specific key used to encrypt that message, it is useless.

8.3 OPERATIONS USED BY ENCRYPTION ALGORITHMS

Although the methods of encryption/decryption have changed dramatically since the advent of computers, there are still only two basic operations that can be carried out on a piece of plaintext: substitution and transposition. The only real difference is that, earlier these were carried out with the alphabet, nowadays they are carried out on binary bits.

Substitution

Substitution operations replace bits in the plaintext with other bits decided upon by the algorithm, to produce ciphertext. This substitution then just has to be reversed to produce plaintext from ciphertext. This can be made increasingly complicated. For instance one plaintext character could correspond to one of a number of ciphertext characters (homophonic substitution), or each character of plaintext is substituted by a character of corresponding position in a length of another text (running cipher).

Example 8.4 Julius Caesar was one of the first to use substitution encryption to send messages to troops during the war. The substitution method he invented advances each character three spaces in the alphabet. Thus,

THIS IS SUBSTITUTION CIPHER (8.1)

becomes

WKLV LU VXEVWL WXWLRQ FLSKHU. (8.2)

Transposition

Transposition (or permutation) does not alter any of the bits in plaintext, but instead moves their positions around within it. If the resultant ciphertext is then put through more transpositions, the end result is increasing security.

XOR

XOR is an exclusive-or operation. It is a Boolean operator such that if 1 of two bits is true, then so is the result, but if both are true **or** both are false then the result is false. For example,

$$\begin{aligned} 0 \text{ XOR } 0 &= 0 \\ 1 \text{ XOR } 0 &= 1 \\ 0 \text{ XOR } 1 &= 1 \\ 1 \text{ XOR } 1 &= 0 \end{aligned} \tag{8.3}$$

A surprising amount of commercial software uses simple XOR functions to provide security, including the USA digital cellular telephone network and many office applications, and it is trivial to crack. However the XOR operation, as will be seen later in this paper, is a vital part of many advanced Cryptographic Algorithms when performed between long blocks of bits that also undergo substitution and/or transposition.

8.4 SYMMETRIC (SECRET KEY) CRYPTOGRAPHY

Symmetric Algorithms (or **Single Key Algorithms** or **Secret Key Algorithms**) have one key that is used both to encrypt and decrypt the message, hence the name. In order for the recipient to decrypt the message they need to have an identical copy of the key. This presents one major problem, the distribution of the keys. Unless the recipient can meet the sender in person and obtain a key, the key itself must be transmitted to the recipient, and is thus susceptible to eavesdropping. However, single key algorithms are fast and efficient, especially if large volumes of data need to be processed.

In Symmetric Cryptography, the two parties that exchange messages use the same algorithm. Only the key is changed from time to time. The same plaintext with a different key results in a different ciphertext. The encryption algorithm is available to the public, hence should be strong and well-tested. The more powerful the algorithm, the less likely that an attacker will be able to decrypt the resulting cipher.

The size of the key is critical in producing strong ciphertext. The US National Security Agency, NSA stated in the mid-1990s that a 40-bit length was acceptable to them (i.e., they

could crack it sufficiently quickly!). Increasing processor speeds, combined with loosely-coupled multi-processor configurations, have brought the ability to crack such short keys within the reach of potential hackers. In 1998, it was suggested that in order to be strong, the key size needs to be at least 56 bits long. It was argued by an expert group as early as 1996 that 90 bits is a more appropriate length. Today, the most secure schemes use 128-bit keys or even longer keys.

Symmetric Cryptography provides a means of satisfying the requirement of message content security, because the content cannot be read without the secret key. There remains a risk of exposure, however, because neither party can be sure that the other party has not exposed the secret key to a third party (whether accidentally or intentionally).

Symmetric Cryptography can also be used to address integrity and authentication requirements. The sender creates a summary of the message, or **Message Authentication Code (MAC)**, encrypts it with the secret key, and sends that with the message. The recipient then re-creates the MAC, decrypts the MAC that was sent, and compares the two. If they are identical, then the message that was received must have been identical with that which was sent.

As mentioned earlier, a major difficulty with symmetric schemes is that the secret key has to be possessed by both parties, and hence has to be transmitted from whoever creates it to the other party. Moreover, if the key is compromised, all of the message transmission security measures are undermined. The steps taken to provide a secure mechanism for creating and passing on the secret key are referred to as **Key Management**.

The technique does not adequately address the non-repudiation requirement, because both parties have the same secret key. Hence each is exposed to the risk of fraudulent falsification of a message by the other, and a claim by either party not to have sent a message is credible, because the other may have compromised the key.

There are two types of Symmetric Algorithms–Block Ciphers and Stream Ciphers.

Definition 8.3 Block Ciphers usually operate on groups of bits called blocks. Each block is processed a multiple number of times. In each round the key is applied in a unique manner. The more the number of iterations, the longer is the encryption process, but results in a more secure ciphertext.

Definition 8.4 Stream Ciphers operate on plaintext one bit at a time. Plaintext is streamed as raw bits through the encryption algorithm. While a block cipher will produce the same ciphertext from the same plaintext using the same key, a stream cipher will not. The ciphertext produced by a stream cipher will vary under the same conditions.

How long should a key be? There is no single answer to this question. It depends on the specific situation. To determine how much security one needs, the following questions must be answered:

(i) What is the worth of the data to be protected?

(ii) How long does it need to be secure?
(iii) What are the resources available to the cryptanalyst/hacker?

A customer list might be worth Rs 1000, an advertisement data might be worth Rs. 50,000 and the master key for a digital cash system might be worth millions. In the world of stock markets, the secrets have to be kept for a couple of minutes. In the newspaper business today's secret is tomorrow's headlines. The census data of a country have to be kept secret for months (if not years). Corporate trade secrets are interesting to rival companies and military secrets are interesting to rival militaries. Thus, the security requirements can be specified in these terms. For example, one may require that the key length must be such that there is a probability of 0.0001% that a hacker with the resources of Rs 1 million could break the system in 1 year, assuming that the technology advances at a rate of 25% per annum over that period. The minimum key requirement for different applications are listed in Table 8.1. This table should be used as a guideline only.

Table 8.1 *Minimum key requirements for different applications*

Type of information	*Lifetime*	*Minimum key length*
Tactical military information	Minutes/hours	56–64 bits
Product announcements	Days/weeks	64 bits
Interest rates	Days/weeks	64 bits
Trade secrets	decades	112 bits
Nuclear bomb secrets	> 50 years	128 bits
Identities of spies	> 50 years	128 bits
Personal affairs	> 60 years	> 128 bits
Diplomatic embarrassments	> 70 years	> 128 bits

Future computing power is difficult to estimate. A rule of thumb is that the efficiency of computing equipment divided by price doubles every 18 months, and increases by a factor of 10 every five years. Thus, in 50 years the fastest computer will be 10 billion times faster than today's! These numbers refer to general-purpose computers. We cannot predict what kind of specialized crypto-system breaking computers might be developed in the years to come.

Two symmetric algorithms, both block ciphers, will be discussed in this chapter. These are the **Data Encryption Standard** (DES) and the **International Data Encryption Algorithm** (IDEA).

8.5 DATA ENCRYPTION STANDARD (DES)

DES, an acronym for the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS) 46-3, which describes the Data Encryption Algorithm (DEA). The DEA is also defined in the ANSI standard X9.32.

Created by IBM, DES came about due to a public request by the US National Bureau of Standards (NSB) requesting proposals for a Standard Cryptographic Algorithm that satisfied the following criteria:

(i) Provides a high level of security
(ii) The security depends on keys, not the secrecy of the algorithm
(iii) The security is capable of being evaluated
(iv) The algorithm is completely specified and easy to understand
(v) It is efficient to use and adaptable
(vi) Must be available to all users
(vii) Must be exportable

DEA is essentially an improvement of the 'Algorithm Lucifer' developed by IBM in the early 1970s. The US National Bureau of Standards published the Data Encryption Standard in 1975. While the algorithm was basically designed by IBM, the NSA and NBS (now NIST) played a substantial role in the final stages of the development. The DES has been extensively studied since its publication and is the best known and the most widely used Symmetric Algorithm in the world.

The DEA has a 64-bit block size and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). The DEA is a Symmetric Cryptosystem, specifically a 16-round Feistel Cipher and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message, or to generate and verify a Message Authentication Code (MAC). The DEA can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography provides an ideal solution to this problem.

NIST re-certifies DES (FIPS 46-1, 46-2, 46-3) every five years. FIPS 46-3 reaffirms DES usage as of October 1999, but single DES is permitted only for legacy systems. FIPS 46-3 includes a definition of triple-DES (TDEA, corresponding to X9.52). Within a few years, DES and triple-DES will be replaced with the Advanced Encryption Standard.

DES has now been in world-wide use for over 20 years, and due to the fact that it is a defined standard means that any system implementing DES can communicate with any other system using it. DES is used in banks and businesses all over the world, as well as in networks (as Kerberos) and to protect the password file on UNIX Operating Systems (as CRYPT).

DES Encryption

DES is a symmetric, block-cipher algorithm with a key length of 64 bits, and a block size of 64 bits (i.e. the algorithm operates on successive 64 bit blocks of plaintext). Being symmetric, the same key is used for encryption and decryption, and DES also uses the same algorithm for encryption and decryption.

First a transposition is carried out according to a set table (the initial permutation), the 64-bit plaintext block is then split into two 32-bit blocks, and 16 identical operations called rounds are carried out on each half. The two halves are then joined back together, and the reverse of the

initial permutation carried out. The purpose of the first transposition is not clear, as it does not affect the security of the algorithm, but is thought to be for the purpose of allowing plaintext and ciphertext to be loaded into 8-bit chips in byte-sized pieces.

In any round, only one half of the original 64-bit block is operated on. The rounds alternate between the two halves. One round in DES consists of the following.

Key Transformation

The 64-bit key is reduced to 56 by removing every eighth bit (these are sometimes used for error checking). Sixteen different 48-bit subkeys are then created – one for each round. This is achieved by splitting the 56-bit key into two halves, and then circularly shifting them left by 1 or 2 bits, depending on the round. After this, 48 of the bits are selected. Because they are shifted, different groups of key bits are used in each subkey. This process is called a compression permutation due to the transposition of the bits and the reduction of the overall size.

Expansion Permutation

After the key transformation, whichever half of the block is being operated on undergoes an expansion permutation. In this operation, the expansion and transposition are achieved simultaneously by allowing the 1st and 4th bits in each 4 bit block to appear twice in the output, i.e., the 4th input bit becomes the 5th and 7th output bits (see Fig. 8.2).

The expansion permutation achieves 3 things: Firstly it increases the size of the half-block from 32 bits to 48, the same number of bits as in the compressed key subset, which is important as the next operation is to XOR the two together. Secondly, it produces a longer string of data for the substitution operation that subsequently compresses it. Thirdly, and most importantly, because in the subsequent substitutions the 1st and 4th bits appear in two S-boxes (described shortly), they affect two substitutions. The effect of this is that the dependency of the output bits on the input bits increases rapidly, and so, therefore, does the security of the algorithm.

Fig. 8.2 *The Expansion Permutation.*

XOR

The resulting 48-bit block is then XORed with the appropriate subset key for that round.

Substitution

The next operation is to perform substitutions on the expanded block. There are eight substitution boxes, called S-boxes. The first S-box operates on the first 6 bits of the 48-bit

expanded block, the 2nd S-box on the next six, and so on. Each S-box operates from a table of 4 rows and 16 columns, each entry in the table is a 4-bit number. The 6-bit number the S-box takes as input is used to look up the appropriate entry in the table in the following way. The 1st and 6th bits are combined to form a 2-bit number corresponding to a row number, and the 2nd to 5th bits are combined to form a 4-bit number corresponding to a particular column. The net result of the substitution phase is eight 4-bit blocks that are then combined into a 32-bit block.

It is the non-linear relationship of the S-boxes that really provide DES with its security, all the other processes within the DES algorithm are linear, and as such relatively easy to analyze.

Fig. 8.3 *The S-box Substitution.*

Permutation

The 32-bit output of the substitution phase then undergoes a straightforward transposition using a table sometimes known as the P-box.

After all the rounds have been completed, the two 'half-blocks' of 32 bits are recombined to form a 64-bit output, the final permutation is performed on it, and the resulting 64-bit block is the DES encrypted ciphertext of the input plaintext block.

DES Decryption

Decrypting DES is very easy (if one has the correct key!). Thanks to its design, the decryption algorithm is identical to the encryption algorithm–the only alteration that is made, is that to decrypt DES ciphertext, the subsets of the key used in each round are used in reverse, i.e., the 16th subset is used first.

Security of DES

Unfortunately, with advances in the field of cryptanalysis and the huge increase in available computing power, DES is no longer considered to be very secure. There are algorithms that can be used to reduce the number of keys that need to be checked, but even using a straightforward brute-force attack and just trying every single possible key, there are computers that can crack DES in a matter of minutes. It is rumoured that the US National Security Agency (NSA) can crack a DES encrypted message in 3–15 minutes.

If a time limit of 2 hours to crack a DES encrypted file is set, then you have to check all possible keys (2^{56}) in two hours, which is roughly 5 trillion keys per second. While this may

seem like a huge number, consider that a $10 Application-Specific Integrated Circuits (ASICs) chip can test 200 million keys per second, and many of these can be paralleled together. It is suggested that a $10 million investment in ASICs would allow a computer to be built that would be capable of breaking a DES encrypted message in 6 minutes.

DES can no longer be considered a sufficiently secure algorithm. If a DES-encrypted message can be broken in minutes by supercomputers today, then the rapidly increasing power of computers means that it will be a trivial matter to break DES encryption in the future (when a message encrypted today may still need to be secure). An extension of DES called DESX is considered to be virtually immune to an exhaustive key search.

8.6 INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

IDEA was created in its first form by Xuejia Lai and James Massey in 1990, and was called the Proposed Encryption Standard (PES). In 1991, Lai and Massey strengthened the algorithm against differential cryptanalysis and called the result Improved PES (IPES). The name of IPES was changed to International Data Encryption Algorithm (IDEA) in 1992. IDEA is perhaps best known for its implementation in PGP (Pretty Good Privacy).

The Algorithm

IDEA is a symmetric, block-cipher algorithm with a key length of 128 bits, a block size of 64 bits, and as with DES, the same algorithm provides encryption and decryption.

IDEA consists of 8 rounds using 52 subkeys. Each round uses six subkeys, with the remaining four being used for the output transformation. The subkeys are created as follows.

Firstly the 128-bit key is divided into eight 16-bit keys to provide the first eight subkeys. The bits of the original key are then shifted 25 bits to the left, and then it is again split into eight subkeys. This shifting and then splitting is repeated until all 52 subkeys (SK1-SK52) have been created.

The 64-bit plaintext block is first split into four blocks (B1–B4). A round then consists of the following steps (OB stands for output block):

OB1 = B1 * SK1 (multiply 1st sub-block with 1st subkey)
OB2 = B2 + SK2 (add 2nd sub-block to 2nd subkey)
OB3 = B3 + SK3 (add 3rd sub-block to 3rd subkey)
OB4 = B4 * SK4 (multiply 4th sub-block with 4th subkey)
OB5 = OB1 XOR OB3 (XOR results of steps 1 and 3)
OB6 = OB2 XOR OB4
OB7 = OB5 * SK5 (multiply result of step 5 with 5th subkey)
OB8 = OB6 + OB7 (add results of steps 5 and 7)
OB9 = OB8 * SK6 (multiply result of step 8 with 6th subkey)

OB10 = OB7 + OB9
OB11 = OB1 XOR OB9 (XOR results of steps 1 and 9)
OB12 = OB3 XOR OB9
OB13 = OB2 XOR OB10
OB14 = OB4 XOR OB10

The input to the next round, is the four sub-blocks OB11, OB13, OB12, OB14 in that order.

After the eighth round, the four final output blocks (F1–F4) are used in a final transformation to produce four sub-blocks of ciphertext (C1–C4) that are then rejoined to form the final 64-bit block of ciphertext.

C1 = F1 * SK49
C2 = F2 + SK50
C3 = F3 + SK51
C4 = F4 * SK52
Ciphertext = C1 C2 C3 C4.

Security Provided by IDEA

Not only is IDEA approximately twice as fast as DES, but it is also considerably more secure. Using a brute-force approach, there are 2^{128} possible keys. If a billion chips that could each test 1 billion keys a second were used to try and crack an IDEA-encrypted message, it would take them 10^{13} years which is considerably longer than the age of the universe! Being a fairly new algorithm, it is possible a better attack than brute-force will be found, which, when coupled with much more powerful machines in the future may be able to crack a message. However, for a long way into the future, IDEA seems to be an extremely secure cipher.

8.7 RC CIPHERS

The RC ciphers were designed by Ron Rivest for the RSA Data Security. RC stands for *Ron's Code* or *Rivest Cipher*. **RC2** was designed as a quick-fix replacement for DES that is more secure. It is a block cipher with a variable key size that has a propriety algorithm. RC2 is a variable-key-length cipher. However, when using the Microsoft Base Cryptographic Provider, the key length is hard-coded to 40 bits. When using the Microsoft Enhanced Cryptographic Provider, the key length is 128 bits by default and can be in the range of 40 to 128 bits in 8-bit increments.

RC4 was developed by Ron Rivest in 1987. It is a variable-key-size stream cipher. The details of the algorithm have not been officially published. The algorithm is extremely easy to describe and program. Just like RC2, 40-bit RC4 is supported by the Microsoft Base Cryptographic provider, and the Enhanced provider allows keys in the range of 40 to 128 bits in 8-bit increments.

RC5 is a block cipher designed for speed. The block size, key size and the number of iterations are all variables. In particular, the key size can be as large as 2,048 bits.

All the encryption techniques discussed so far belong to the class of symmetric cryptography (DES, IDEA and RC Ciphers). We now look at the class of Asymmetric Cryptographic Techniques.

8.8 ASYMMETRIC (PUBLIC-KEY) ALGORITHMS

Public-key Algorithms are asymmetric, that is to say the key that is used to encrypt the message is different from the key used to decrypt the message. The encryption key, known as the public key is used to encrypt a message, but the message can only be decoded by the person that has the decryption key, known as the private key.

This type of algorithm has a number of advantages over traditional symmetric ciphers. It means that the recipient can make their public key widely available – anyone wanting to send them a message uses the algorithm and the recipient's public key to do so. An eavesdropper may have both the algorithm and the public key, but will still not be able to decrypt the message. Only the recipient, with their private key can decrypt the message.

A disadvantage of public-key algorithms is that they are more computationally intensive than symmetric algorithms, and therefore encryption and decryption take longer. This may not be significant for a short text message, but certainly is for long messages or audio/video.

The **Public-Key Cryptography Standards** (PKCS) are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography. First published in 1991 as a result of meetings with a small group of early adopters of public-key technology, the PKCS documents have become widely referenced and implemented. Contributions from the PKCS series have become part of many formal and de facto standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL.

The next two sections describe two popular public-key algorithms, the RSA Algorithm and the Pretty Good Privacy (PGP) Hybrid Algorithm.

8.9 THE RSA ALGORITHM

RSA, named after its three creators–Rivest, Shamir and Adleman, was the first effective public-key algorithm, and for years has withstood intense scrutiny by cryptanalysts all over the world. Unlike symmetric key algorithms, where, as long as one presumes that an algorithm is not flawed, the security relies on having to try all possible keys, public-key algorithms rely on it being computationally unfeasible to recover the private key from the public key.

RSA relies on the fact that it is easy to multiply two large prime numbers together, but extremely hard (i.e. time consuming) to factor them back from the result. Factoring a number means finding its prime factors, which are the prime numbers that need to be multiplied together in order to produce that number. For example,

$$10 = 2 \times 5$$
$$60 = 2 \times 2 \times 3 \times 5$$
$$2^{113} - 1 = 3391 \times 23279 \times 65993 \times 1868569 \times 1066818132868207$$

The algorithm

Two very large prime numbers, normally of equal length, are randomly chosen then multiplied together.

$$N = A \times B \tag{8.4}$$
$$T = (A - 1) \times (B - 1) \tag{8.5}$$

A third number is then also chosen randomly as the public key (E) such that it has no common factors (i.e. is relatively prime) with T. The private key (D) is then:

$$D = E^{-1} \bmod T \tag{8.6}$$

To encrypt a block of plaintext (M) into ciphertext (C):

$$C = M^E \bmod N \tag{8.7}$$

To decrypt:

$$M = C^D \bmod N \tag{8.8}$$

Example 8.5 **Consider the following implementation of the RSA algorithm.**

1st prime (A) = 37

2nd prime (B) = 23

So,

$$N = 37 \times 23 = 851$$
$$T = (37 - 1) \times (23 - 1) = 36 \times 23 = 792$$

E must have no factors other than 1 in common with 792.

E (public key) could be 5.

D (private key) = $5^{-1} \bmod 792 = 317$

To encrypt a message (M) of the character 'G':

If G is represented as 7 (7th letter in alphabet), then $M = 7$.

C (ciphertext) = $7^5 \bmod 851 = 638$

To decrypt: $M = 638^{317} \bmod 851 = 7$.

Security of RSA

The security of RSA algorithm depends on the ability of the hacker to factorize numbers. New, faster and better methods for factoring numbers are constantly being devised. The current best for long numbers is the *Number Field Sieve.* Prime Numbers of a length that was unimaginable a mere decade ago are now factored easily. Obviously the longer a number is, the harder it is to factor, and so the better the security of RSA. As theory and computers improve, larger and

larger keys will have to be used. The disadvantage in using extremely long keys is the computational overhead involved in encryption/decryption. This will only become a problem if a new factoring technique emerges that requires keys of such lengths to be used that necessary key length increases much faster than the increasing average speed of computers utilising the RSA algorithm.

In 1997, a specific assessment of the security of 512-bit RSA keys showed that one may be factored for less than $1,000,000 in cost and eight months of effort. It is therefore believed that 512-bit keys provide insufficient security for anything other than short-term needs. RSA Laboratories currently recommend key sizes of 768 bits for personal use, 1024 bits for corporate use, and 2048 bits for extremely valuable keys like the root-key pair used by a certifying authority. Security can be increased by changing a user's keys regularly and it is typical for a user's key to expire after two years (the opportunity to change keys also allows for a longer length key to be chosen).

Even without using huge keys, RSA is about 1000 times slower to encrypt/decrypt than DES. This has resulted in it not being widely used as a stand-alone cryptography system. However, it is used in many **hybrid cryptosystems** such as PGP. The basic principle of hybrid systems is to encrypt plaintext with a Symmetric Algorithm (usually DES or IDEA); the symmetric algorithm's key is then itself encrypted with a public-key algorithm such as RSA. The RSA encrypted key and symmetric algorithm-encrypted message are then sent to the recipient, who uses his private RSA key to decrypt the Symmetric Algorithm's key, and then that key to decrypt the message. This is considerably faster than using RSA throughout, and allows a different symmetric key to be used each time, considerably enhancing the security of the Symmetric Algorithm.

RSA's future security relies solely on advances in factoring techniques. Barring an astronomical increase in the efficiency of factoring techniques, or available computing power, the 2048-bit key will ensure very secure protection into the foreseeable future. For instance an Intel Paragon, which can achieve 50,000 mips (million operations per second), would take a million years to factor a 2048-bit key using current techniques.

8.10 PRETTY GOOD PRIVACY (PGP)

Pretty Good Privacy (PGP) is a hybrid cryptosystem that was created by Phil Zimmerman and released onto the Internet as a freeware program in 1991. PGP is not a new algorithm in its own right, but rather a series of other algorithms that are performed along with a sophisticated protocol. PGP's intended use was for e-mail security, but there is no reason why the basic principles behind it could not be applied to any type of transmission.

PGP and its source code is freely available on the Internet. This means that since its creation PGP has been subjected to an enormous amount of scrutiny by cryptanalysts, who have yet to find an exploitable fault in it.

PGP has four main modules: a symmetric cipher – IDEA for message encryption, a public key algorithm–RSA to encrypt the IDEA key and hash values, a one-way hash function–MD5 for signing, and a random number generator.

The fact that the body of the message is encrypted with a symmetric algorithm (IDEA) means that PGP generated e-mails are a lot faster to encrypt and decrypt than ones using simple RSA. The key for the IDEA module is randomly generated each time as a one-off session key, this makes PGP very secure, as even if one message was cracked, all previous and subsequent messages would remain secure. This session key is then encrypted with the public key of the recipient using RSA. Given that keys up to 2048 bits long can be used, this is extremely secure. MD5 can be used to produce a hash of the message, which can then be signed by the sender's private key. Another feature of PGP's security is that the user's private key is encrypted using a hashed pass-phrase rather than simply a password, making the private key extremely resistant to copying even with access to the user's computer.

Generating true random numbers on a computer is notoriously hard. PGP tries to achieve randomness by making use of the keyboard latency when the user is typing. This means that the program measures the gap of time between each key-press. Whilst at first this may seem to be distinctly non-random, it is actually fairly effective-people take longer to hit some keys than others, pause for thought, make mistakes and vary their overall typing speed on all sorts of factors such as knowledge of the subject and tiredness. These measurements are not actually used directly but used to trigger a pseudo-random number generator. There are other ways of generating random numbers, but to be much better than this gets very complex.

PGP uses a very clever, but complex, protocol for key management. Each user generates and distributes their public key. If James is happy that a person's public key belongs to who it claims to belong to, then he can sign that person's public key and James's program will then accept messages from that person as valid. The user can allocate levels of trust to other users. For instance, James may decide that he completely trusts Earl to sign other peoples' keys, in effect saying "his word is good enough for me". This means that if Rachel, who has had her key signed by Earl, wants to communicate with James, she sends James her signed key. James's program recognises Earl's signature, has been told that Earl can be trusted to sign keys, and so accepts Rachel's key as valid. In effect Earl has introduced Rachel to James.

PGP allows many levels of trust to be assigned to people, and this is best illustrated in Fig. 8.4.

The explanations are as follows.

1st line

James has signed the keys of Earl, Sarah, Jacob and Kate. James completely trusts Earl to sign other peoples' keys, does not trust Sarah at all, and partially trusts Jacob and Kate (he trusts Jacob more than Kate).

Fig. 8.4 *An Example of a PGP User Web.*

2nd Line

Although James has not signed Sam's key he still trusts Sam to sign other peoples' keys, may be on Bob's say so or due to them actually meeting. Because Earl has signed Rachel's key, Rachel is validated (but not trusted to sign keys). Even though Bob's key is signed by Sarah and Jacob, because Sarah is not trusted and Jacob only partially trusted, Bob is not validated. Two partially trusted people, Jacob and Kate, have signed Archie's key, therefore Archie is validated.

3rd line

Sam, who is fully trusted, has signed Hal's key, therefore Hal is validated. Louise's key has been signed by Rachel and Bob, neither of whom is trusted, therefore Louise is not validated.

Odd one out

Mike's key has not been signed by anyone in James' group, maybe James found it on the Internet and does not know whether it is genuine or not.

PGP never prevents the user from sending or receiving e-mail, it does however warn the user if a key is not validated, and the decision is then up to the user as to whether to heed the warning or not.

Key Revocation

If a user's private key is compromised then they can send out a key revocation certificate. Unfortunately this does not guarantee that everyone with that user's public key will receive it, as keys are often swapped in a disorganised manner. Additionally, if the user no longer has the private key then they cannot issue a certificate, as the key is required to sign it.

Security of PGP

"A chain is only as strong as its weakest link" is the saying and it holds true for PGP. If the user chooses a 40-bit RSA key to encrypt his session keys and never validates any users, then PGP will not be very secure. If however a 2048-bit RSA key is chosen and the user is reasonably vigilant, then PGP is the closest thing to military-grade encryption the public can hope to get their hands on.

The Deputy Director of the NSA was quoted as saying:

"If all the personal computers in the world, an estimated 260 million, were put to work on a single PGP-encrypted message, it would still take an estimated 12 million times the age of the universe, on average, to break a single message."

A disadvantage of public-key cryptography is that anyone can send you a message using your public key, it is then necessary to prove that this message came from who it claims to have been sent by. A message encrypted by someone's private key, can be decrypted by anyone with their public key. This means that if the sender encrypted a message with his private key, and then encrypted the resulting ciphertext with the recipient's public key, the recipient would be able to decrypt the message with first their private key, and then the sender's public key, thus recovering the message and proving it came from the correct sender.

This process is very time-consuming, and therefore rarely used. A much more common method of digitally signing a message is using a method called One-Way Hashing.

8.11 ONE-WAY HASHING

A **One-Way Hash Function** is a mathematical function that takes a message string of any length (pre-string) and returns a smaller fixed-length string (hash value). These functions are designed in such a way that not only is it very difficult to deduce the message from its hashed version, but also that even given that all hashes are a certain length, it is extremely hard to find two messages that hash to the same value. In fact to find two messages with the same hash from a 128-bit hash function, 2^{64} hashes would have to be tried. In other words, the hash value of a file is a small unique 'fingerprint'. Even a slight change in an input string should cause the hash value to change drastically. Even if 1 bit is flipped in the input string, at least half of the bits in the hash value will flip as a result. This is called an **Avalanche Effect**.

H= hash value, f= hash function, M= original message/pre-string, then

$$H = f(M). \tag{8.9}$$

If you know M then H is easy to compute. However knowing H and f, it is not easy to compute M, and is hopefully computationally unfeasible.

As long as there is a low risk of collision (i.e. 2 messages hashing to the same value), and the hash is very hard to reverse, then a one-way hash function proves extremely useful for a number of aspects of cryptography.

If you one-way hash a message, the result will be a much shorter but still unique (at least statistically) number. This can be used as proof of ownership of a message without having to reveal the contents of the actual message. For instance rather than keeping a database of copyrighted documents, if just the hash values of each document were stored, then not only would this save a lot of space, but it would also provide a great deal of security. If copyright then needs to be proved, the owner could produce the original document and prove it hashes to that value.

Hash-functions can also be used to prove that no changes have been made to a file, as adding even one character to a file would completely change its hash value.

By far the most common use of hash functions is to digitally sign messages. The sender performs a one-way hash on the plaintext message, encrypts it with his private key and then encrypts both with the recipient's public key and sends in the usual way. On decrypting the ciphertext, the recipient can use the sender's public key to decrypt the hash value, he can then perform a one-way hash himself on the plaintext message, and check this with the one he has received. If the hash values are identical, the recipient knows not only that the message came from the correct sender, as it used their private key to encrypt the hash, but also that the plaintext message is completely authentic as it hashes to the same value.

The above method is greatly preferable to encrypting the whole message with a private key, as the hash of a message will normally be considerably smaller than the message itself. This means that it will not significantly slow down the decryption process in the same way that decrypting the entire message with the sender's public key, and then decrypting it again with the recipient's private key would. The PGP system uses the MD5 hash function for precisely this purpose.

The Microsoft Cryptographic Providers support three hash algorithms: MD4, MD5 and SHA. Both MD4 and MD5 were invented by Ron Rivest. MD stands for **Message Digest**. Both algorithms produce 128-bit hash values. MD5 is an improved version of MD4. SHA stands for Secure Hash Algorithm. It was designed by NIST and NSA. SHA produces 160-bit hash values, longer than MD4 and MD5. SHA is generally considered more secure than other algorithms and is the recommended hash algorithm.

8.12 OTHER TECHNIQUES

One Time Pads

The one-time pad was invented by Major Joseph Mauborgne and Gilbert Bernam in 1917, and is an unconditionally secure (i.e. unbreakable) algorithm. The theory behind a one-time pad is simple. The pad is a non-repeating random string of letters. Each letter on the pad is used once only to encrypt one corresponding plaintext character. After use, the pad must *never* be re-used. As long as the pad remains secure, so is the message. This is because a random key added to a non-random message produces completely random ciphertext, and there is absolutely no amount of analysis or computation that can alter that. If both pads are destroyed then the original message will never be recovered. There are two major drawbacks:

Firstly, it is extremely hard to generate truly random numbers, and a pad that has even a couple of non-random properties is theoretically breakable. Secondly, because the pad can never be reused no matter how large it is, the length of the pad must be the same as the length of the message which is fine for text, but virtually impossible for video.

Steganography

Steganography is not actually a method of encrypting messages, but hiding them within something else to enable them to pass undetected. Traditionally this was achieved with invisible ink, microfilm or taking the first letter from each word of a message. This is now achieved by hiding the message within a graphics or sound file. For instance in a 256-greyscale image, if the least significant bit of each byte is replaced with a bit from the message then the result will be indistinguishable to the human eye. An eavesdropper will not even realise a message is being sent. This is not cryptography however, and although it would fool a human, a computer would be able to detect this very quickly and reproduce the original message.

Secure Mail and S/MIME

Secure Multipurpose Internet Mail Extensions (S/MIME) is a *de facto* standard developed by RSA Data Security, Inc., for sending secure mail based on public-key cryptography. MIME is the industry standard format for electronic mail, which defines the structure of the message's body. S/MIME-supporting e-mail applications add digital signatures and encryption capabilities to that format to ensure message integrity, data origin authentication and confidentiality of electronic mail.

When a signed message is sent, a detached signature in the PKCS #7 format is sent along with the message as an attachment. The signature attachment contains the hash of the original message signed with the sender's private key, as well as the signer certificate. S/MIME also supports messages that are first signed with the sender's private key and then enveloped using the recipients' public keys.

8.13 SECURE COMMUNICATION USING CHAOS FUNCTIONS

Chaos functions have also been used for secure communications and cryptographic applications. The implication of a chaos function here is an iterative difference equation that exhibits chaotic behaviour. If we observe the fact that cryptography has more to do with unpredictability rather than randomness, chaos functions are a good choice because of their property of *unpredictability*. If a hacker intercepts part of the sequence, he will have no information on how to predict what comes next. The unpredictability of chaos functions makes them a good choice for generating the keys for symmetric cryptography.

Example 8.6 Consider the difference equation

$$x_{n+1} = ax_n(1 - x_n) \tag{8.10}$$

For $a = 4$, this function behaves like a chaos function, i.e.,

(i) the values obtained by successive iterations are unpredictable, and

(ii) the function is extremely sensitive to the initial condition, x_0.

For any given initial condition, this function will generate values of x_n between 0 and 1 for each iteration. These values are good candidates for *key* generation. In single-key cryptography, a key is used for enciphering the message. This key is usually a pseudo noise (PN) sequence. The message can be simply XORed with the key in order to scramble it. Since x_n takes positive values that are always less than unity, the binary equivalent of these fractions can serve as keys. Thus, one of the ways of generating keys from these random, unpredictable decimal numbers is to directly use their binary representation. The lengths of these binary sequences will be limited only by the accuracy of the decimal numbers, and hence very long binary keys can be generated. The recipient must know the initial condition in order to generate the keys for decryption.

For application in single key cryptography the following two factors need to be decided

(i) The start value for the iterations (x_0), and

(ii) The number for decimal places of the mantissa that are to be supported by the calculating machine (to avoid round off error).

For single-key cryptography, the chaos values obtained after some number of iteration are converted to binary fractions whose first 64 bits are taken to generate PN sequences. These initial iterations would make it still more difficult for the hacker to guess the initial condition. The starting value should be taken between 0 and 1. A good choice of the starting value can improve the performance slightly.

The secrecy of the starting number, x_0, is the key to the success of this algorithm. Since chaos functions are extremely sensitive to even errors of 10^{-30} in the starting number (x_0), it means that we can have 10^{30} unique starting combinations. Therefore, a hacker who knows the chaos function and the encryption algorithm has to try out 10^{30} different start combinations. In the DES algorithm the hacker had to try out approximately 10^{19} different key values.

Chaos based algorithms require a high computational overhead to generate the chaos values as well as high computational speeds. Hence, it might not be suitable for bulk data encryption.

8.14 CRYPTANALYSIS

Cryptanalysis is the science (or black art!) of recovering the plaintext of a message from the ciphertext without access to the key. In cryptanalysis, it is always assumed that the cryptanalyst

has full access to the algorithm. An attempted cryptanalysis is known as an attack, of which there are five major types:

- *Brute force attack*: This technique requires a large amount of computing power and a large amount of time to run. It consists of trying all possibilities in a logical manner until the correct one is found. For the majority of encryption algorithms a brute force attack is impractical due to the large number of possibilities.
- *Ciphertext-only*: The only information the cryptanalyst has to work with is the ciphertext of various messages all encrypted with the same algorithm.
- *Known-plaintext*: In this scenario, the cryptanalyst has access not only to the ciphertext of various messages, but also the corresponding plaintext as well.
- *Chosen-plaintext*: The cryptanalyst has access to the same information as in a known plaintext attack, but this time may choose the plaintext that gets encrypted. This attack is more powerful, as specific plaintext blocks can be chosen that may yield more information about the key. An adaptive-chosen-plaintext attack is merely one where the cryptanalyst may repeatedly encrypt plaintext, thereby modifying the input based on the results of a previous encryption.
- *Chosen-ciphertext*: The cryptanalyst uses a relatively new technique called differential cryptanalysis, which is an interactive and iterative process. It works through many rounds using the results from previous rounds, until the key is identified. The cryptanalyst repeatedly chooses ciphertext to be decrypted, and has access to the resulting plaintext. From this they try to deduce the key.

There is only one totally secure algorithm, the one-time pad. All other algorithms can be broken given infinite time and resources. Modern cryptography relies on making it computationally unfeasible to break an algorithm. This means, that while it is theoretically possible, the time scale and resources involved make it completely unrealistic.

If an algorithm is presumed to be perfect, then the only method of breaking it relies on trying every possible key combination until the resulting ciphertext makes sense. As mentioned above, this type of attack is called a brute-force attack. The field of parallel computing is perfectly suited to the task of brute force attacks, as every processor can be given a number of possible keys to try, and they do not need to interact with each other at all except to announce the result. A technique that is becoming increasingly popular is parallel processing using thousands of individual computers connected to the Internet. This is known as distributed computing. Many cryptographers believe that brute force attacks are basically ineffective when long keys are used. An encryption algorithm with a large key (over 100 bits) can take millions of years to crack, even with powerful, networked computers of today. Besides, adding a single extra key doubles the cost of performing a brute force cryptanalysis.

Regarding brute force attack, there are a couple of other pertinent questions. What if the original plaintext is itself a cipher? In that case, how will the hacker know if he has found the right key. In addition, is the cryptanalyst sitting at the computer and watching the result of each

key that is being tested? Thus, we can assume that brute force attack is impossible provided long enough keys are being used.

Here are some of the techniques that have been used by cryptanalysts to attack ciphertext.

- *Differential cryptanalysis*: As mentioned before, this technique uses an iterative process to evaluate cipher that has been generated using an iterative block algorithm (e.g. DES). Related plaintext is encrypted using the same key. The difference is analysed. This technique proved successful against DES and some hash functions.
- *Linear Cryptanalysis*: In this, pairs of plaintext and ciphertext are analysed and a linear approximation technique is used to determine the behaviour of the block cipher. This technique was also used successfully against DES.
- *Algebraic attack*: This technique exploits the mathematical structure in block ciphers. If the structure exists, a single encryption with one key might produce the same result as a double encryption with two different keys. Thus the search time can be reduced.

However, strong or weak the algorithm used to encrypt it, a message can be thought of as secure if the time and/or resources needed to recover the plaintext greatly exceed the benefits bestowed by having the contents. This could be because the cost involved is greater than the financial value of the message, or simply that by the time the plaintext is recovered the contents will be outdated.

8.15 POLITICS OF CRYPTOGRAPHY

Widespread use of cryptosystems is something most governments are not particularly happy about–precisely because it threatens to give more privacy to the individual, including criminals. For many years, police forces have been able to tap phone lines and intercept mail, however, in an encrypted future that may become impossible.

This has led to some strange decisions on the part of governments, particularly the United States government. In the United States, cryptography is classified as a munition and the export of programs containing cryptosystems is tightly controlled. In 1992, the Software Publishers Association reached agreement with the State Department to allow the export of software that contained RSA's RC2 and RC4 encryption algorithms, but only if the key size was limited to 40 bits as opposed to the 128 bit keys available for use within the US. This significantly reduced the level of privacy produced. In 1993 the US Congress had asked the National Research Council to study US cryptographic policy. Its 1996 report, the result of two years' work, offered the following conclusions and recommendations:

- "On balance, the advantages of more widespread use of cryptography outweigh the disadvantages."
- "No law should bar the manufacture, sale or use of any form of encryption within the United States."
- "Export controls on cryptography should be progressively relaxed but not eliminated."

In 1997 the limit on the key size was increased to 56 bits. The US government has proposed several methods whereby it would allow the export of stronger encryption, all based on a system where the US government could gain access to the keys if necessary, for example the *clipper* chip. Recently there has been a lot of protest from the cryptographic community against the US government imposing restrictions on the development of cryptographic techniques. The article by Ronald L. Rivest, Professor, MIT, in the October 1998 issue of the *Scientific American*, (pages 116–117) titled "The Case against Regulating Encryption Technology," is an example of such a protest. The resolution of this issue is regarded to be one of the most important for the future of e-commerce.

8.16 CONCLUDING REMARKS

In this section we present a brief history of cryptography. People have tried to conceal information in written form since writing was developed. Examples survive in stone inscriptions and papyruses showing that many ancient civilizations including the Egyptians, Hebrews and Assyrians all developed cryptographic systems. The first recorded use of cryptography for correspondence was by the Spartans who (as early as 400 BC) employed a cipher device called a *scytale* to send secret communications between military commanders.

The scytale consisted of a tapered baton around which was wrapped a piece of parchment inscribed with the message. Once unwrapped the parchment appeared to contain an incomprehensible set of letters, however when wrapped around another baton of identical size the original text appears.

The Greeks were therefore the inventors of the first transposition cipher and in the fourth century BC the earliest treatise on the subject was written by a Greek, Aeneas Tacticus, as part of a work entitled *On the Defence of Fortifications*. Another Greek, Polybius, later devised a means of encoding letters into pairs of symbols using a device known as the *Polybius checkerboard* which contains many elements common to later encryption systems. In addition to the Greeks there are similar examples of primitive substitution or transposition ciphers in use by other civilizations including the Romans. The Polybius checkerboard consists of a five by five grid containing all the letters of the alphabet. Each letter is converted into two numbers, the first is the row in which the letter can be found and the second is the column. Hence the letter A becomes 11, the letter B 12 and so forth.

The Arabs were the first people to clearly understand the principles of cryptography. They devised and used both substitution and transposition ciphers and discovered the use of letter frequency distributions in cryptanalysis. As a result of this, by approximately 1412, al-Kalkashandi could include in his encyclopaedia *Subh al-a'sha* a respectable, if elementary, treatment of several cryptographic systems. He also gave explicit instructions on how to cryptanalyze ciphertext using letter frequency counts including examples illustrating the technique.

European cryptography dates from the Middle Ages during which it was developed by the Papal and Italian city states. The earliest ciphers involved only vowel substitution (leaving the consonants unchanged). Circa 1379 the first European manual on cryptography, consisting of a compilation of ciphers, was produced by Gabriele de Lavinde of Parma, who served Pope Clement VII. This manual contains a set of keys for correspondents and uses symbols for letters and nulls with several two character code equivalents for words and names. The first brief code vocabularies, called nomenclators, were expanded gradually and for several centuries were the mainstay of diplomatic communication for nearly all European governments. In 1470 Leon Battista Alberti described the first cipher disk in *Trattati in cifra* and the *Traicté de chiffres*, published in 1586 by Blaise de Vigernère contained a square table commonly attributed to him as well as descriptions of the first plaintext and ciphertext autokey systems.

By 1860 large codes were in common use for diplomatic communications and cipher systems had become a rarity for this application. However, cipher systems prevailed for military communications (except for high-command communication because of the difficulty of protecting codebooks from capture or compromise). During the US Civil War the Federal Army extensively used transposition ciphers. The Confederate Army primarily used the Vigenère cipher and on occasional monoalphabetic substitution. While the Union cryptanalysts solved most of the intercepted Confederate ciphers, the Confederacy, in desperation, sometimes published Union ciphers in newspapers, appealing for help from readers in cryptanalysing them.

During the first world war both sides employed cipher systems almost exclusively for tactical communication while code systems were still used mainly for high-command and diplomatic communication. Although field cipher systems such as the US Signal Corps cipher disk lacked sophistication, some complicated cipher systems were used for high-level communications by the end of the war. The most famous of these was the German ADFGVX fractionation cipher.

In the 1920s the maturing of mechanical and electromechanical technology came together with the needs of telegraphy and radio to bring about a revolution in cryptodevices–the development of rotor cipher machines. The concept of the rotor had been anticipated in the older mechanical cipher disks however it was an American, Edward Hebern, who recognised that by hardwiring a monoalphabetic substitution in the connections from the contacts on one side of an electrical rotor to those on the other side and cascading a collection of such rotors, polyalphabetic substitutions of almost any complexity could be produced. From 1921 and continuing through the next decade, Hebern constructed a series of steadily improving rotor machines that were evaluated by the US Navy. It was undoubtedly this work which led to the United States' superior position in cryptology during the Second World War. At almost the same time as Hebern was inventing the rotor cipher machine in the United States, European engineers such as Hugo Koch (Netherlands) and Arthur Scherbius (Germany) independently discovered the rotor concept and designed the precursors to the most famous cipher machine in history, the German Enigma Machine which was used during World War 2. These machines

were also the stimulus for the TYPEX, the cipher machine employed by the British during World War 2.

The United States introduced the M-134-C (SIGABA) cipher machine during World War 2. The Japanese cipher machines of World War 2 have an interesting history linking them to both the Hebern and the Enigma machines. After Herbert Yardley, an American cryptographer who organised and directed the US government's first formal code-breaking efforts during and after the first world war, published *The American Black Chamber* in which he outlined details of the American successes in cryptanalysing the Japanese ciphers, the Japanese government set out to develop the best cryptomachines possible. With this in mind, it purchased the rotor machines of Hebern and the commercial Enigmas, as well as several other contemporary machines, for study. In 1930 the Japanese's first rotor machine, code named RED by US cryptanalysts, was put into service by the Japanese Foreign Office. However, drawing on experience gained from cryptanalysing the ciphers produced by the Hebern rotor machines, the US Army Signal Intelligence Service team of cryptanalysts succeeded in cryptanalysing the RED ciphers. In 1939, the Japanese introduced a new cipher machine, code-named PURPLE by US cryptanalysts, in which the rotors were replaced by telephone stepping switches. The greatest triumphs of cryptanalysis occurred during the Second World War when the Polish and British cracked the Enigma ciphers and the American cryptanalysts broke the Japanese RED, ORANGE and PURPLE ciphers. These developments played a major role in the Allies' conduct of World War 2.

After World War 2 the electronics that had been developed in support of radar were adapted to cryptomachines. The first electrical cryptomachines were little more than rotor machines where the rotors had been replaced by electronic substitutions. The only advantage of these electronic rotor machines was their speed of operation as they were still affected by the inherent weaknesses of the mechanical rotor machines.

The era of computers and electronics has meant an unprecedented freedom for cipher designers to use elaborate designs which would be far too prone to error if handled with pencil and paper, or far too expensive to implement in the form of an electromechanical cipher machine. The main thrust of development has been in the development of block ciphers, beginning with the *LUCIFER* project at IBM, a direct ancestor of the *DES* (Data Encryption Standard).

There is a place for both symmetric and public-key algorithms in modern cryptography. Hybrid cryptosystems successfully combine aspects of both and seem to be secure and fast. While PGP and its complex protocols are designed with the Internet community in mind, it should be obvious that the encryption behind it is very strong and could be adapted to suit many applications. There may still be instances when a simple algorithm is necessary, and with the security provided by algorithms like IDEA, there is absolutely no reason to think of these as significantly less secure.

An article posted on the Internet on the subject of picking locks stated: "The most effective door opening tool in any burglars' toolkit remains the crowbar". This also applies to cryptanalysis – direct action is often the most effective. It is all very well transmitting your messages with 128-bit IDEA encryption, but if all that is necessary to obtain that key is to walk up to one of the computers used for encryption with a floppy disk, then the whole point of encryption is negated. In other words, an incredibly strong algorithm is not sufficient. For a system to be effective there must be effective management protocols involved. Finally, in the words of Sir Edgar Allen Poe, "Human ingenuity cannot concoct a cipher which human ingenuity cannot resolve."

SUMMARY

- A cryptosystem is a collection of algorithms and associated procedures for hiding and revealing information. Cryptanalysis is the process of analysing a cryptosystem, either to verify its integrity or to break it for ulterior motives. An attacker is a person or system that performs cryptanalysis in order to break a cryptosystem. The process of attacking a cryptosystem is often called cracking. The job of the cryptanalyst is to find the weaknesses in the cryptosystem.
- A message being sent is known as plaintext. The message is coded using a cryptographic algorithm. This process is called encryption. An encrypted message is known as ciphertext, and is turned back into plaintext by the process of decryption.
- A key is a value that causes a cryptographic algorithm to run in a specific manner and produce a specific ciphertext as an output. The key size is usually measured in bits. The bigger the key size, the more secure will be the algorithm.
- Symmetric algorithms (or single key algorithms or secret key algorithms) have one key that is used both to encrypt and decrypt the message, hence their name. In order for the recipient to decrypt the message they need to have an identical copy of the key. This presents one major problem, the distribution of the keys.
- Block ciphers usually operate on groups of bits called blocks. Each block is processed a multiple number of times. In each round the key is applied in a unique manner. The more the number of iterations, the longer is the encryption process, but results in a more secure ciphertext.
- Stream ciphers operate on plaintext one bit at a time. Plaintext is streamed as raw bits through the encryption algorithm. While a block cipher will produce the same ciphertext from the same plaintext using the same key, a stream cipher will not. The ciphertext produced by a stream cipher will vary under the same conditions.
- To determine how much security one needs, the following questions must be answered:
  1. What is the worth of the data to be protected?
  2. How long does it need to be secure?
  3. What are the resources available to the cryptanalyst/hacker?

- Two symmetric algorithms, both block ciphers, were discussed in this chapter. These are the Data Encryption Standard (DES) and the International Data Encryption Algorithm (IDEA).
- Public-key algorithms are asymmetric, that is to say the key that is used to encrypt the message is different to the key used to decrypt the message. The encryption key, known as the public key is used to encrypt a message, but the message can only be decoded by the person that has the decryption key, known as the private key. Rivest, Shamir and Adleman (RSA) algorithm and the Pretty Good Privacy (PGP) are two popular public-key encryption techniques.
- RSA relies on the fact that it is easy to multiply two large prime numbers together, but extremely hard (i.e. time consuming) to factor them back from the result. Factoring a number means finding its prime factors, which are the prime numbers that need to be multiplied together in order to produce that number.
- A one-way hash function is a mathematical function that takes a message string of any length (pre-string) and returns a smaller fixed-length string (hash value). These functions are designed in such a way that not only is it very difficult to deduce the message from its hashed version, but also that even given that all hashes are a certain length, it is extremely hard to find two messages that hash to the same value.
- Chaos functions can be used for secure communication and cryptographic applications. The chaotic functions are primarily used for generating keys that are essentially unpredictable.
- An attempted unauthorised cryptanalysis is known as an attack, of which there are five major types: Brute force attack, Ciphertext-only, Known-plaintext, Chosen-plaintext and Chosen ciphertext.
- The common techniques that are used by cryptanalysts to attack ciphertext are differential cryptanalysis, linear cryptanalysis and algebraic attack.
- Widespread use of cryptosystems is something most governments are not particularly happy about, because it threatens to give more privacy to the individual, including criminals.

> *Imagination is more important than knowledge.*
>
> *Einstein, Albert (1879-1955)*

PROBLEMS

8.1 We want to test the security of *character* + *x* encrypting technique in which each alphabet of the plaintext is shifted by *n* to produce the ciphertext.

(a) How many different attempts must be made to crack this code assuming brute force attack is being used?

(b) Assuming it takes a computer 1 ms to check out one value of the shift, how soon can this code be broken into?

8.2 Suppose a group of N people want to use secret key cryptography. Each pair of people in the group should be able to communicate secretly. How many distinct keys are required?

8.3 Transposition Ciphers rearrange the letters of the plaintext without changing the letters themselves. For example, a very simple transposition cipher is the *rail fence*, in which the plaintext is staggered between two rows and then read off to give the ciphertext. In a two row rail fence the message MERCHANT TAYLORS' SCHOOL becomes:

M R H N T Y O S C O L
E C A T A L R S H O

Which is read out as: MRHNTYOSCOLECATALRSHO.

(a) If a cryptanalyst wants to break into the rail fence cipher, how many distinct attacks must he make, given the length of the ciphertext is n?

(b) Suggest a decrypting algorithm for the rail fence cipher.

8.4 One of the most famous field ciphers ever was a fractionation system – the *ADFGVX cipher*, which was employed by the German Army during the first world war. This system was so named because it used a 6×6 matrix to substitution-encrypt the 26 letters of the alphabet and 10 digits into pairs of the symbols A, D, F, G, V and X. The resulting biliteral cipher is only an intermediate cipher, it is then written into a rectangular matrix and transposed to produce the final cipher which is the one which would be transmitted. Here is an example of enciphering the phrase "Merchant Taylors" with this cipher using the key word "Subject".

	A	**D**	**F**	**G**	**V**	**X**
A	S	U	B	J	E	C
D	T	A	D	F	G	H
F	I	K	L	M	N	O
G	P	Q	R	V	W	X
V	Y	Z	0	1	2	3
X	4	5	6	7	8	9

Plaintext: M E R C H A N T T A Y L O R S
Ciphertext: FG AV GF AX DX DD FV DA DA DD VA FF FX GF AA

This intermediate ciphertext can then be put in a transposition matrix based on a different key.

C	**I**	**P**	**H**	**E**	**R**
1	**4**	**5**	**3**	**2**	**6**
F	G	A	V	G	F
A	X	D	X	D	D
F	V	D	A	D	A
D	D	V	A	F	F
F	X	G	F	A	A

The final cipher is therefore: FAFDFGDDFAVXAAFGXVDXADDVGFDAFA.

(a) If a cryptanalyst wants to break into ADFGVX cipher, how many distinct attacks must he make, given the length of the ciphertext is n?

(b) Suggest a decrypting algorithm for the ADFGVX cipher.

8.5 Consider the knapsack technique for encryption proposed by Ralph Merkle of XEROX and Martin Hellman of Stanford University in 1976. They suggested using the knapsack, or subset-sum, problem as the basis for a public key cryptosystem. This problem entails determining whether a number can be expressed as a sum of some subset of a given sequence of numbers and, more importantly, which subset has the desired sum.

Given a sequence of numbers A, where $A = (a_1 \ldots a_n)$, and a number C, the knapsack problem is to find a subset of $a_1 \ldots a_n$ which sums to C.

Consider the following example:

$$n = 5,\ C = 14,\ A = (1, 10, 5, 22, 3)$$

Solution $= 14 = 1 + 10 + 3$

In general, all the possible sums of all subsets can be expressed by:

$m_1a_1 + m_2a_2 + m_3a_3 + \ldots + m_na_n$ where each m_i is either 0 or 1.

The solution is therefore a *binary vector* $M = (1, 1, 0, 0, 1)$.

There is a total number 2^n of such vectors (in this example $2^5 = 32$)

Obviously not all values of C can be formed from the sum of a subset and some can be formed in more than one way. For example, when $A = (14, 28, 56, 82, 90, 132, 197, 284, 341, 455, 515)$ the figure 515 can be formed in three different ways but the number 516 cannot be formed in any way.

(a) If a cryptanalyst wants to break into this knapsack cipher, how many distinct attacks must he make?

(b) Suggest a decrypting algorithm for the knapsack cipher.

8.6 (a) Use the prime numbers 29 and 61 to generate keys using the RSA Algorithm.

(b) Represent the letters 'RSA' in ASCII and encode them using the key generated above.

(c) Next, generate keys using the pair of primes, 37 and 67. Which is more secure, the keys in part (a) or part (c)? Why?

COMPUTER PROBLEMS

8.7 Write a program that performs encryption using DES.

8.8 Write a program to encode and decode using IDEA. Compare the number of computations required to encrypt a plaintext using the same keysize for DES and IDEA.

8.9 Write a general program that can factorize a given number.

8.10 Write a program to encode and decode using the RSA algorithm. Plot the number of floating point operations required to be performed by the program versus the key-size.

8.11 Consider the difference equation

$$x_{n+1} = ax_n(1 - x_n)$$

For $a = 4$, this function behaves like a chaos function.

(a) Plot a sequence of 100 values obtained by iterative application of the difference equation. What happens if the starting values $x_0 = 0.5$?

(b) Take two initial conditions (i.e., two different starting values, x_{01} and x_{02}) which are separated by Δx. Use the difference equation to iterate each starting point n times and obtain the final values y_{01} and y_{02}, which are separated by Δy. For a given Δx, plot Δy versus n.

(c) For a given value of n (say $n = 500$), plot Δx verus Δy.

(d) Repeat parts (a), (b) and (c) for $a = 3.7$ and $a = 3.9$. Compare and comment.

(e) Develop a chaos-based encryption program that generates keys for single-key encryption. Use the chaos function

$$x_{n+1} = 4x_n(1-x_n)$$

(f) Compare the encryption speed of this chaos-based program with that of IDEA for a key length of 128 bits.

(g) Compare the security of this chaos-based algorithm with that of IDEA for the 128 bit long key.

Index

A Mathematical Theory of Communication 41
  a scytale 265
AC coefficients 40
Additive White Gaussian Noise (AWGN) 56
Aeneas Tacticus 265
Algebraic attack 264
Asymmetric (Public-Key) Algorithms 254
Asymmetric Encryption 244
  attacker 241
Augmented Generating Function 175
  authenticity 242
Automatic Repeat Request 97
Avalanche Effect 259
Average Conditional Entropy 15
Average Conditional Self-Information 12
Average Mutual Information 11, 14
  average number of nearest neighbours 221
Average Self-Information 11

Bandwidth Efficiency Diagram 60
Binary Entropy Function 12
Binary Golay Code 124
Binary Symmetric Channel 8
Blaise de Vigernère 266
Block Ciphers 247
Block Code 53, 77
Block Codes 53
Block Length 78, 161
Blocklength 161, 168
Brute force attack 263
BSC 13
Burst Error Correction 121
Burst Errors 121

Capacity Boundary 61
  catastrophic 185
Catastrophic Convolutional Code 169
Catastrophic Error Propagation 170
Channel 49
Channel Capacity 50
Channel Coding 48, 76
Channel Coding Theorem 53
Channel Decoder 52
Channel Encoder 52
Channel Formatting 52
Channel Models 48
  channel state information 229
  channel transition probabilities 9