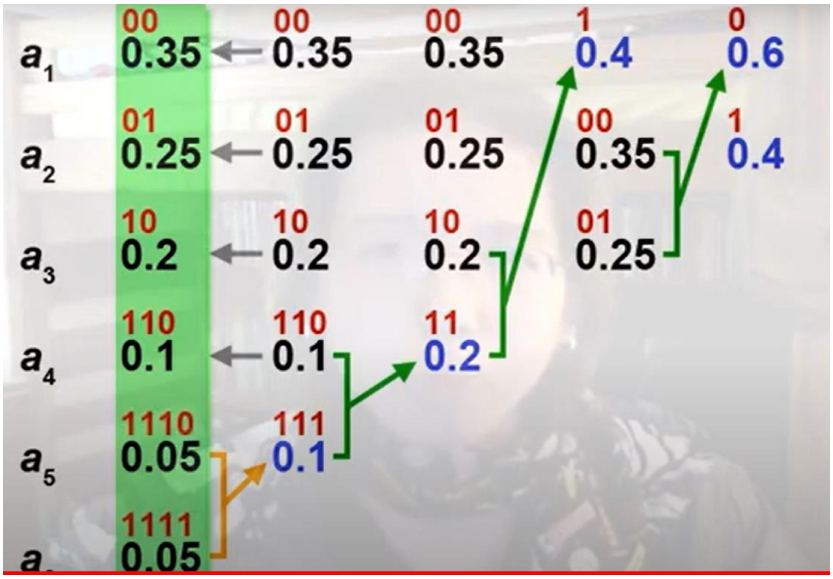
EXPERIMENT NO. 8

Aim: To understand Source Coding Theorem. Study of Huffman coding.
Objective: Study of Huffman coding Encoder and Decoder.
Software Used: MATLAB 7.6 software

Huffman coding is a lossless data compression algorithm. In this algorithm, a variable-length code is assigned to input different characters. The code length is related to how frequently characters are used. Most frequent characters have the smallest codes and longer codes for least frequent characters.

Huffman coding is an entropy coding algorithm used for lossless data compression, developed by David Huffman in 1952. The idea is to assign variable-length codes to input characters, the most frequent character gets the smallest code and the least frequent character gets the largest code. The Huffman algorithm is a so-called "greedy" approach to solving this problem in the sense that at each step, the algorithm chooses the best available option. It turns out that this is sufficient for finding the best encoding.

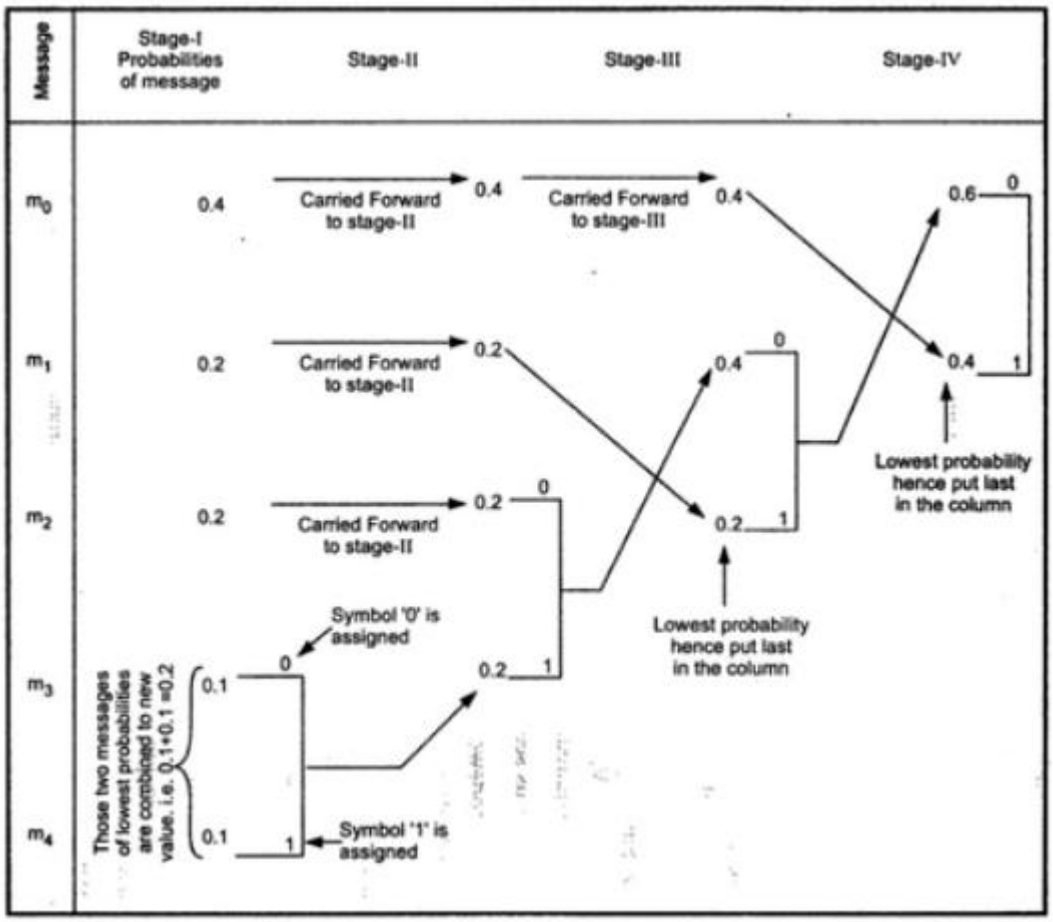


Message	Probability	Digits obtained by tracing	Codeword obtained by reading digits of column-3 from LSB side	No. of digits
m_0	$p_0 = 0.4$	1	1	(1)
m_1	$p_1 = 0.2$	10	01	(2)
m_2	$p_2 = 0.2$	000	000	(3)
m_3	$p_3 = 0.1$	0100	0010	(4)
m_4	$p_4 = 0.1$	1100	0011	(4)

Consider that the source generates five messages $m_0, m_1, ... m_4$. The probabilities of these messages are as shown in 2nd column of Table 1.8.3.

1. The messages are arranged according to their decreasing probabilities. For example m_3 and m_4 have lowest probabilities and hence they are put at the bottom in column of stage-I.
2. The two messages of lowest probabilities are assigned binary '0' and '1'.
3. The two lowest probabilities in stage-I are added. Observe that the sum of two probabilities is 0.1 + 0.1 = 0.2.
4. The sum of probabilities in stage-I is placed in stage-II such that the probabilities are in descending order. Observe that 0.2 is placed last in stage-II.
5. Now the last two probabilities are assigned '0' to '1' and they are added. Thus the sum of last two probabilities in stage-II is 0.2 + 0.2 = 0.4.
6. The sum of last two probabilities (i.e. 0.4) is placed in stage-III such that the probabilities are in descending order. Again '0' and '1' is assigned to the last two probabilities.
7. Similarly the values in stage-IV are obtained. Since there are only two values in stage-IV, these two values are assigned digits 0 and 1 and no further repetition is required.

Now let us see how the codewords for messages are obtained.



We know that equation 1.4.6 that average information per message (entropy) is given as,

$$H = \sum_{k=1}^{M} p_k \log_2\left(\frac{1}{p_k}\right)$$

For five messages above equation can be expanded as,

$$H = p_0 \log_2\left(\frac{1}{p_0}\right) + p_1 \log_2\left(\frac{1}{p_1}\right) + p_2 \log_2\left(\frac{1}{p_2}\right) + p_3 \log_2\left(\frac{1}{p_3}\right) + p_4 \log_2\left(\frac{1}{p_4}\right)$$

Here we started from $k = 0$. Putting values of probabilities in above equation from Table 1.8.5 we get,

$$H = 0.4 \log_2\left(\frac{1}{0.4}\right) + 0.2 \log_2\left(\frac{1}{0.2}\right) + 0.2 \log_2\left(\frac{1}{0.2}\right) + 0.1 \log_2\left(\frac{1}{0.1}\right) + 0.1 \log_2\left(\frac{1}{0.1}\right)$$

$$= 0.52877 + 0.46439 + 0.46439 + 0.33219 + 0.33219$$

$$= 2.12193 \text{ bits of information / message} \quad \ldots (1.8.8)$$

Now let us calculate the average number of binary digits (binits) per message. Since each message is coded with different number of binits, we should use their probabilities to calculate average number of binary digits (binits) per message. It is calculated as follows :

$$\text{Average number of binary digits per message} = \sum \left(\begin{matrix}\text{Probability of} \\ \text{message}\end{matrix}\right) \times \left(\begin{matrix}\text{No. of digits} \\ \text{in codeword}\end{matrix}\right)$$

$$= (0.4 \times 1) + (0.2 \times 2) + (0.2 \times 3) + (0.1 \times 4) + (0.1 \times 4)$$

$$= 2.2 \text{ binary digits / message} \quad \ldots (1.8.9)$$

Thus it is clear from equation 1.8.8 and equation 1.8.9 that Huffman coding assigns binary digits to each message such that Average number of binary digits per message are nearly equal to average bits of information per message (i.e. H). This means because of Huffman coding one binary digit carries almost one bit of information, which is the maximum information that can be conveyed by one digit.

Advantages of Huffman Coding

- Huffman coding is very efficient. It can compress data very quickly and effectively.
- Huffman coding is relatively simple to implement.
- Huffman coding can be used with any type of data.
- Huffman coding is very effective at compressing large files.

Flowchart:

Algorithm:

Program:

Input,

Output,

Result:

Questions:

Solve the following two numerical:

1. Apply Huffman Coding for the symbols [A E H N G S] generated by a DMS with probabilities [0.19,0.15,0.2,0.16,0.4,0.08] Also calculate coding efficiency.
2. Encode the following symbols using Huffman source coding technique and calculate coding efficiency [1/4,1/8,1/16, 1/16, 1/16,1/4,1/16,1/8]

Conclusion:

Timely Submission(10)	Journal Presentation(10)	Performance (10)	Understanding(10)	Oral(10)	Total (50)
Sub Teacher Sign					

EXPERIMENT NO. 9

Aim: To understand linear block coding and decoding techniques.

Objective: Study of Linear Block Code Encoder and Decoder.

Software Used: MATLAB 7.6 software

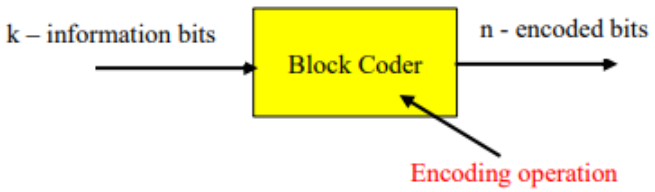
Theory:

Errors are introduced in the data through the channel. The channel noise interferes the signal. The number of errors introduced due to channel noise are minimized by the encoder by adding redundant bits. This increases the overall data rate. Hence channel has to accommodate this increased data rate, and the system becomes slightly complex due to coding techniques. The codes are classified as block or convolution codes.

For a linear code, if two code words are added by modulo 2 arithmetic then, it produces a third code word in the case.

LINEAR BLOCK CODES:

(n, k) Block codes



n-digit codeword made up of k-information digits and (n-k) redundant parity check digits. The rate or efficiency for this code is k/n.

$$Code\ efficiency\ r = \frac{k}{n} = \frac{Number\ of\ information\ bits}{Total\ number\ of\ bits\ in\ codeword}$$

Note: unlike source coding, in which data is compressed, here redundancy is deliberately added, to achieve error detection.

SYSTEMATIC BLOCK CODES

A systematic block code consists of vectors whose 1st k elements (or last k-elements) are identical to the message bits, the remaining (n-k) elements being check bits. A code vector then takes the form:

$$X = (m_0, m_1, m_2, \ldots\ldots m_{k-1}, c_0, c_1, c_2, \ldots\ldots c_{n-k})$$

Or

$$X = (c_0, c_1, c_2, \ldots\ldots c_{n-k}, m_0, m_1, m_2, \ldots\ldots m_{k-1})$$

Systematic code: information digits are explicitly transmitted together with the parity check bits. For the code to be systematic, the k-information bits must be transmitted contiguously as a block, with the parity check bits making up the code word as another contiguous block.

A systematic linear block code will have a generator matrix of the form:

$$G = [P \mid I_k]$$

Systematic codewords are sometimes written so that the message bits occupy the left-hand portion of the codeword and the parity bits occupy the right-hand portion.

Parity check matrix (H)

Will enable us to decode the received vectors. For each (kxn) generator matrix G, there exists an (n-k)xn matrix H, such that rows of G are orthogonal to rows of H i.e., $GH^T = 0$, where H^T is the transpose of H. to fulfil the orthogonal requirements for a systematic code, the components of H matrix are written as:

$$H = [I_{n-k} \mid P^T]$$

In a systematic code, the 1st k-digits of a code word are the data message bits and last (n-k) digits are the parity check bits, formed by linear combinations of message bits $m_0, m_1, m_2, \ldots m_{k-1}$

It can be shown that performance of systematic block codes is identical to that of non-systematic block codes.

A codeword (X) consists of n digits $x_0, x_1, x_2, \ldots x_{n-1}$ and a data word (message word) consists of k digits $m_0, m_1, m_2, \ldots m_{k-1}$

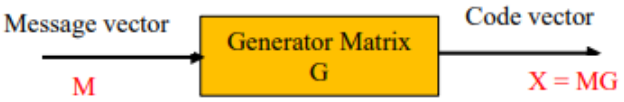
For the general case of linear block codes, all the n digits of X are formed by linear combinations (modulo-2 additions) of k message bits. A special case, where $x_0 = m_0, x_1 = m_1, x_2 = m_2 \ldots x_{k-1} = mk-1$ and the remaining digits from x_{k+1} to x_n are linear combinations of $m_0, m_1, m_2, \ldots m_{k-1}$ is known as a systematic code.

The codes described in this chapter are binary codes, for which the alphabet consists of symbols 0 and 1 only. The encoding and decoding functions involve the binary arithmetic operations of modulo-2 addition and multiplication.

Matrix representation of Block codes

- An (n, k) block code consists of n-bit vectors
- Each vector corresponding to a unique block of k-message bits
- There are 2^k different k-bit message blocks & 2^n possible n-bit vectors
- The fundamental strategy of block coding is to choose the 2^k code vectors such that the minimum distance is as large as possible. In error correction, distance of two words (of same length) plays a fundamental role.

Block codes in which the message bits are transmitted in unaltered form are called systematic code.



For the block of 'K' message bits (n-k) parity bits or check bits are added. Hence the total number of bits at the output of channel encoder are 'n'. Such codes are called (n,k) block codes.

The check bits play the role of error detection and correction. The job of linear block code is to generate those check bits.

The code word can be represented as

$$X=mG$$

Where

X= Code vector of 2^k *n size
m = message vector of 2^k *k size
G= generator matrix of k *n size

$[X]\ 2^k *n = [m]\ 2^k * k\ [G]\ k*n$

The generator matrix depends upon the linear block code used. Generally it is represented as

$$G= [I_k : P_k * (n-k)]\ k*n$$

Where

I_k = k*k identity matrix.
P_k = k(n-k) parity matrix.

The rows of parity matrix should not be same in any case.

Example :

Find the code matrix for a (7,4) code.

Total number of bits, n = 7
Message bits, k = 4
n-k = 3
Total number of message = $2^k = 2^4 = 16$
ie. From 0000 to1111
The parity matrix can be taken as

— $[P]\ k* (n=k) = [P]_{4X3}$

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

$I_k = [I]_{4X4}$

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

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

$$C = mG \quad \begin{bmatrix} 0&0&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \\ 0&0&1&1 \\ 0&1&0&0 \\ 0&1&0&1 \\ 0&1&1&0 \\ 0&1&1&1 \\ 1&0&0&0 \\ 1&0&0&1 \\ 1&0&1&0 \\ 1&0&1&1 \\ 1&1&0&0 \\ 1&1&0&1 \\ 1&1&1&0 \\ 1&1&1&1 \end{bmatrix} \quad \left[\begin{array}{cccc:cccc} 1&1&1&:&1&0&0&0 \\ 0&1&1&:&0&1&0&0 \\ 1&0&1&:&0&0&1&0 \\ 1&1&0&:&0&0&0&1 \end{array}\right]$$

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

Each row of the code matrix represents the transmitted code for the respective message.

Flowchart:

Algorithm:

Program:

Input,

Output,

Result:

Conclusion:

_____

_____

_____

_____

_____

ORAL QUESTIONS

1) What is Hamming weight of a code word?
2) What is Hamming distance, Code rate, Word Length,
3) Define Minimum Hamming Distance, Block length, Constraint Length
4) What is block code, Hamming Code.
5) What are the properties of a linear block code?
6) What is a systematic code, Block code?
7) What is singleton bound, Burst Error?
8) What is parity check matrix?
9) How to generate generator matrix.
10) Types of Error control methods
11) What do you mean by (n, k) code?

Timely Submission(10)	Journal Presenattaion(10)	Performance (10)	Understanding(10)	Oral(10)	Total (50)
Sub Teacher Sign					

EXPERIMENT NO. 10

AIM: To Simulate Study of Convolutional codes

OBJECTIVE: Simulation study of Convolution codes

SOFTWARE: Code block Software, Turbo C++.

THEORY:

This experiment describes the encoder and decoder structures for convolutional codes. The encoder will be represented in many different but equivalent ways. Also, the main decoding strategy for convolutional codes, based on the Viterbi Algorithm, will be described. A firm understanding of convolutional codes is an important prerequisite to the understanding of turbo codes. 2.1 Encoder Structure

A convolutional code introduces redundant bits into the data stream through the use of linear shift registers as shown in Figure 2.1.

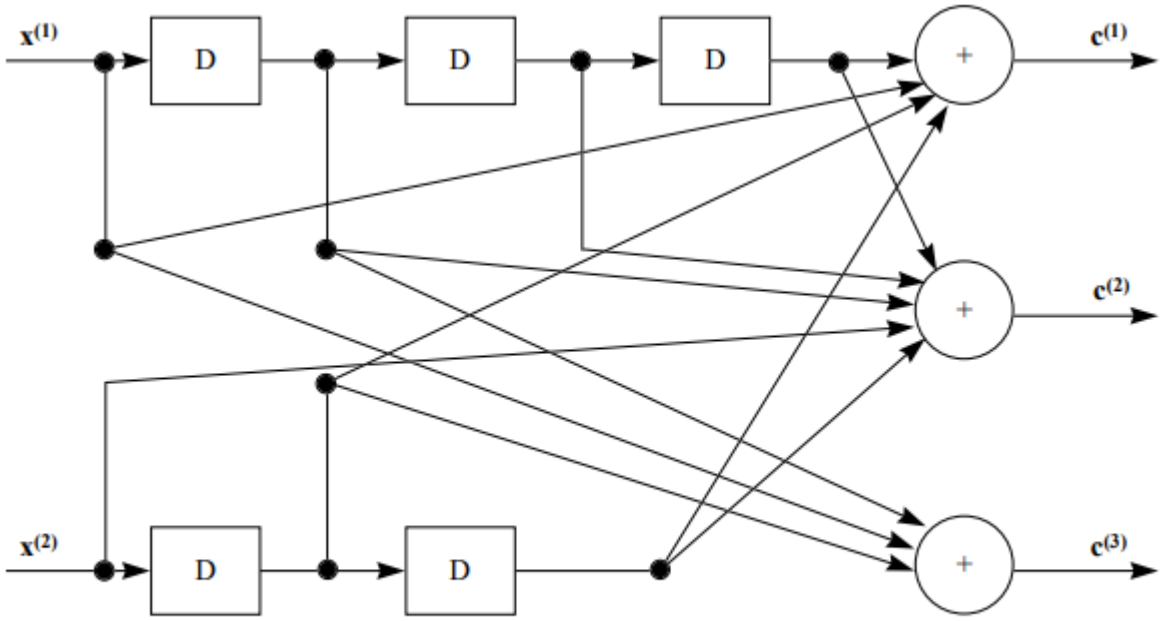


Figure 2.1: Example convolutional encoder where $x^{(i)}$ is an input information bit stream and $c^{(i)}$ is an output encoded bit stream [Wic95].

The information bits are input into shift registers and the output encoded bits are obtained by modulo-2 addition of the input information bits and the contents of the shift registers. The connections to the modulo-2 adders were developed heuristically with no algebraic or combinatorial foundation.

The code rate r for a convolutional code is defined as

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

where k is the number of parallel input information bits and n is the number of parallel output encoded bits at one time interval. The constraint length K for a convolutional code is defined as

$$K = m + 1$$

where m is the maximum number of stages (memory size) in any shift register. The shift registers store the state information of the convolutional encoder and the constraint length relates the number of bits upon which the output depends. For the convolutional encoder shown in Figure 2.1, the code rate r=2/3, the maximum memory size m=3, and the constraint length K=4. A convolutional code can become very complicated with various code rates and constraint lengths. As a result, a simple convolutional code will be used to describe the code properties as shown in Figure 2.2.

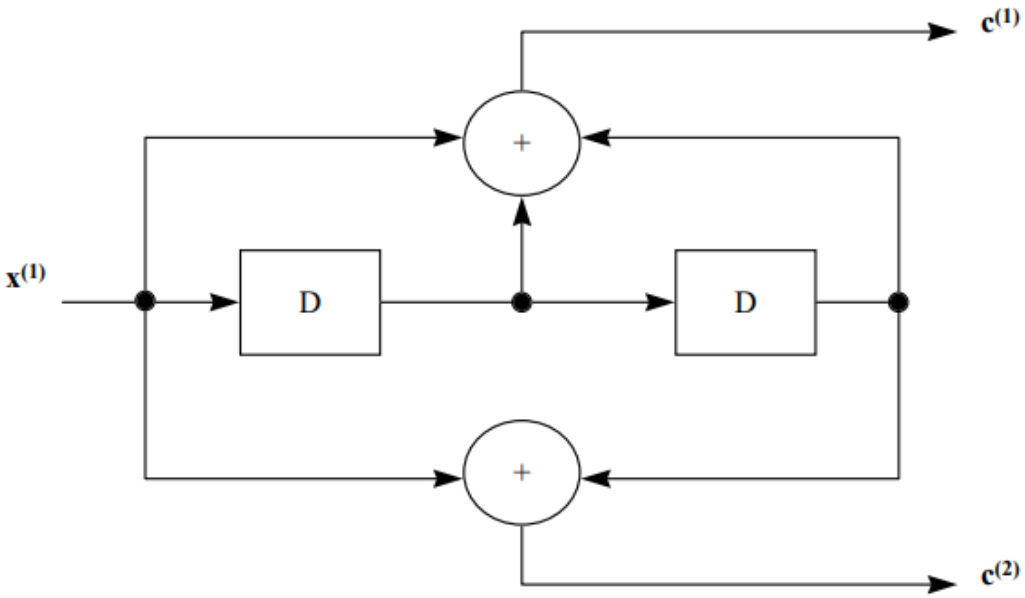


Figure 2.2: Convolutional encoder with k=1, n=2, r=1/2, m=2, and K=3.

Encoder Representations

The encoder can be represented in several different but equivalent ways. They are

1. Generator Representation
2. Tree Diagram Representation
3. State Diagram Representation
4. Trellis Diagram Representation

1. Generator Representation

Generator representation shows the hardware connection of the shift register taps to the modulo-2 adders. A generator vector represents the position of the taps for an output. A "1" represents a connection and a "0" represents no connection. For example, the two generator vectors for the

encoder in Figure 2.2 are g1 = [111] and g2 = [101] where the subscripts 1 and 2 denote the corresponding output terminals.

2 Tree Diagram Representation

The tree diagram representation shows all possible information and encoded sequences for the convolutional encoder. Figure 2.3 shows the tree diagram for the encoder in Figure 2.2 for four input bit intervals.

The tree diagram representation shows all possible information and encoded sequences for the convolutional encoder. Figure 2.3 shows the tree diagram for the encoder in Figure 2.2 for four input bit intervals.

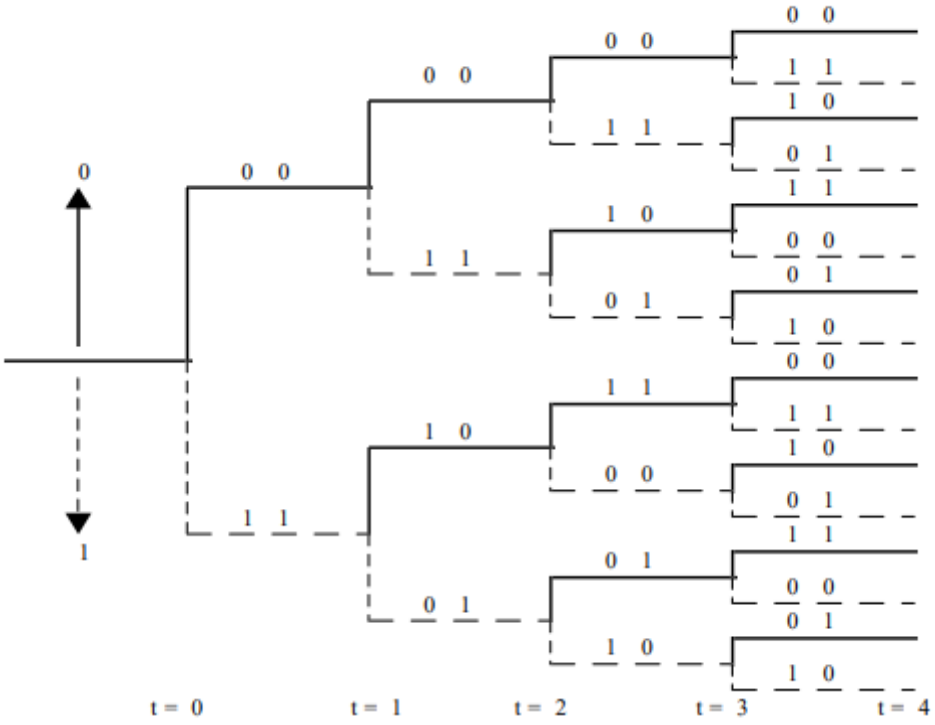


Figure 2.3: Tree diagram representation of the encoder in Figure 2.2 for four input bit intervals.

In the tree diagram, a solid line represents input information bit 0 and a dashed line represents input information bit 1. The corresponding output encoded bits are shown on the branches of the tree. An input information sequence defines a specific path through the tree diagram from left to right. For example, the input information sequence Fu-hua Huang Chapter 2. Convolutional Codes 7 x={1011} produces the output encoded sequence c={11, 10, 00, 01}. Each input information bit corresponds to branching either upward (for input information bit 0) or downward (for input information bit 1) at a tree node.

3 State Diagram Representation The state diagram shows the state information of a convolutional encoder. The state information of a convolutional encoder is stored in the shift registers. Figure 2.4 shows the state diagram of the encoder in Figure 2.2.

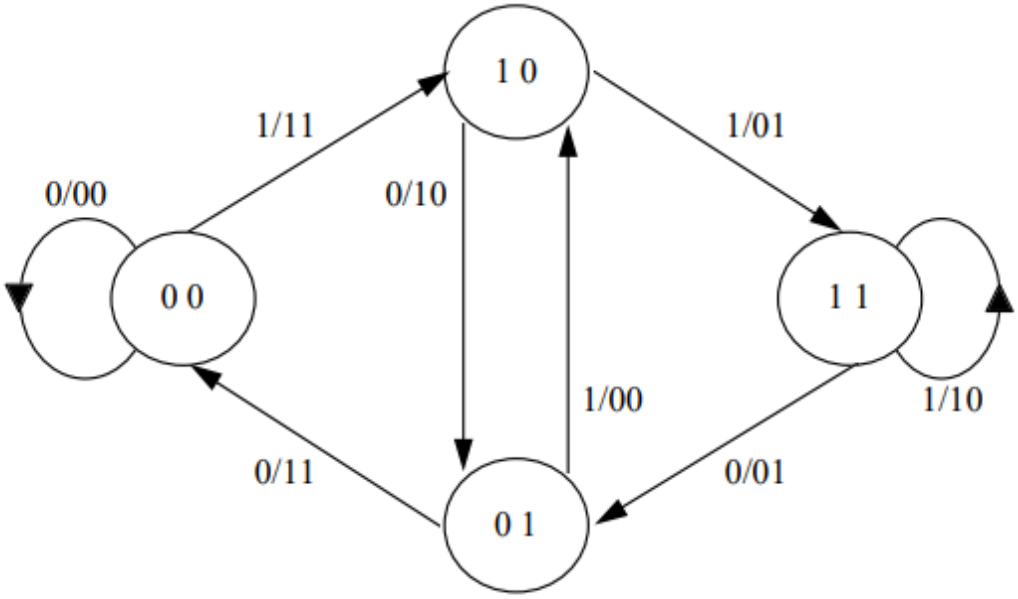


Figure 2.4: State diagram representation of the encoder in Figure 2.2.

In the state diagram, the state information of the encoder is shown in the circles. Each new input information bit causes a transition from one state to another. The path information between the states, denoted as x/c, represents input information bit x and output encoded bits c. It is customary to begin convolutional encoding from the all zero state. For example, the input information sequence x={1011} (begin from the all zero state) leads to the state transition sequence s={10, 01, 10, 11} and produces the output encoded sequence c={11, 10, 00, 01}. Figure 2.5 shows the path taken through the state diagram for the given example.

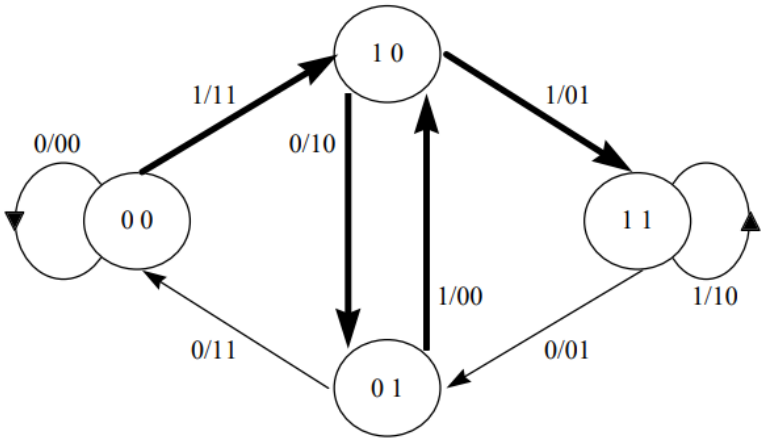


Figure 2.5: The state transitions (path) for input information sequence {1011}.

4 Trellis Diagram Representation

The trellis diagram is basically a redrawing of the state diagram. It shows all possible state transitions at each time step. Frequently, a legend accompanies the trellis diagram to show the state transitions and the corresponding input and output bit mappings (x/c). This compact representation is very helpful for decoding convolutional codes as discussed later. Figure 2.6 shows the trellis diagram for the encoder in Figure 2.2.

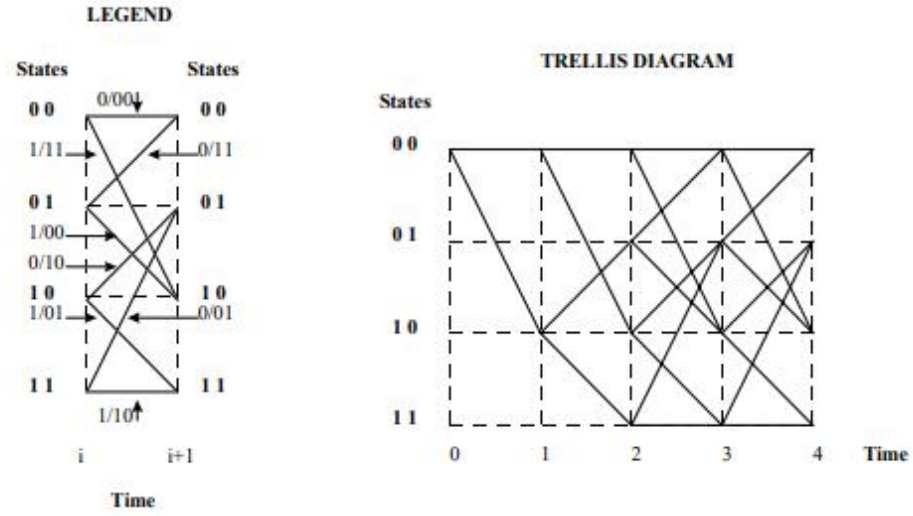


Figure 2.6: Trellis diagram representation of the encoder in Figure 2.2 for four input bit intervals.

Figure 2.7 shows the trellis path for the state transitions in Figure 2.5.

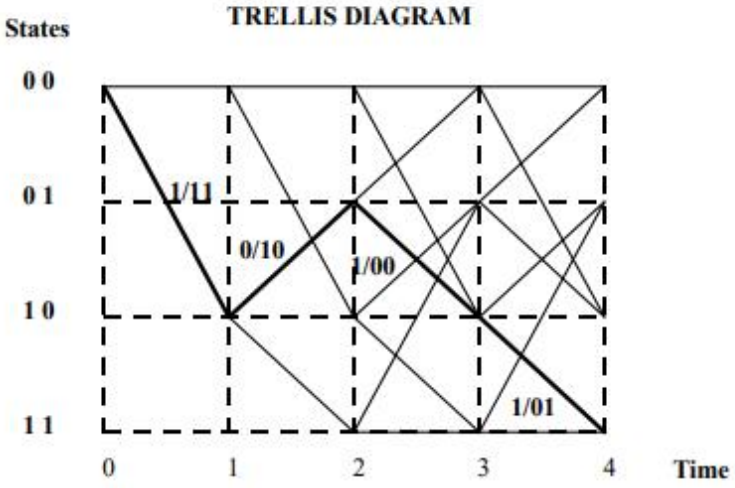


Figure 2.7: Trellis path for the state transitions in Figure 2.5.

Flowchart:

Algorithm:

Program:

Input,

Output,

Result:

Questions:

1. Define code rate related to convolutional codes?
2. Define coding gain and constraint Length?
3. Explain the methods of decoding of convolutional coding?

Conclusion:

Timely Submission(10)	Journal Presenattaion(10)	Performance (10)	Understanding(10)	Oral(10)	Total (50)
Sub Teacher Sign					