Application Of Convolutional Coding In Mb-Ofdm

Sneha R. Mehta
EXTC Final Year
J.D.I.E.T Yavatmal.

Pawan D. Mishra
IT Final year
J.D.I.E.T Yavatmal.

Guided By-

Prof. Athar Ravish Khan
Assistant Professor
J.D.I.E.T Yavatmal.

ABSTRACT

This Paper introduces a procedure for constructing Convolutional codes. A digital transmit-receive system using the convolutional encoder. In this Paper we will concentrate on binary linear time-invariant convolutional. The purpose of a convolutional encoder is to take a single or multi-bit input and generate a matrix of encoded outputs codes.Convolutional encoding is a process of adding redundancy to a signal stream. The Convolution code can be used in a wide variety of error correcting applications.While the implementation of convolutional encoding systems is relatively simple, the process of decoding the resultant data stream at the receiving node can be quite complex. Convolutional codes can be designed to detect or correct the errors. Some convolutional codes available which are used to correct random error and brusts.

1.INTRODUCTION

Convolutional codes represent one technique within the general class of channel codes. Channel codes (also called *error correction codes*) permit reliable communication of an information sequence over a channel that adds noise, introduces bit errors, or otherwise distorts the transmitted signal. Elias introduced convolutional codes in 1955. These codes have found many applications, including deep-space communications and voiceband modems. Convolutional codes continue to play a role in low-latency applications such as speech transmission. Lattice's Convolutional Encoder core is a parameterizable core for convolutional encoding of a continuous input data stream. The core allows variable code rates, constraint lengths and generator polynomials. The core also supports puncturing. Puncturing enables a large range of transmission rates and reduces the bandwidth requirement on the channel. In this Paper we will concentrate on rate-1/n binary linear time-invariant convolutional codes, which are the simplest to understand.

The purpose of a convolutional encoder is to take a single or multi-bit input and generate a matrix of encoded outputs. One reason why this is important is that in digital modulation communications systems (such as wireless communication systems, etc.) noise and other external factors can alter bit sequences. By adding additional bits we make bit error checking more successful and allow for more accurate transfers. By transmitting a greater number of bits than the original signal we introduce a certain redundancy that can be used to determine the original signal in the presence of an error. For our illustration we will assume a 5-bit input and rate-1/2 code (two output bits for every input bit). This will yield a 2x5 output matrix, with the extra bits allowing for the correction.

1.1 Convolutional Encoder and Decoder Basics

Figure 1 shows a digital transmit-receive system using the convolutional coder in MB-OFDM. The digital data stream (such as voice, image or any packetized data) is first convolutionally encoded, then modulated and finally transmitted through a channel. The data received from the channel at the receiver side is first demodulated and then decoded using a Viterbi decoder. The decoded output is equivalent to the original transmitted data stream. Decoding methods are used for decoding the convolutional codes.They are Viterbi algorithm, sequential decoding and feedback decoding.

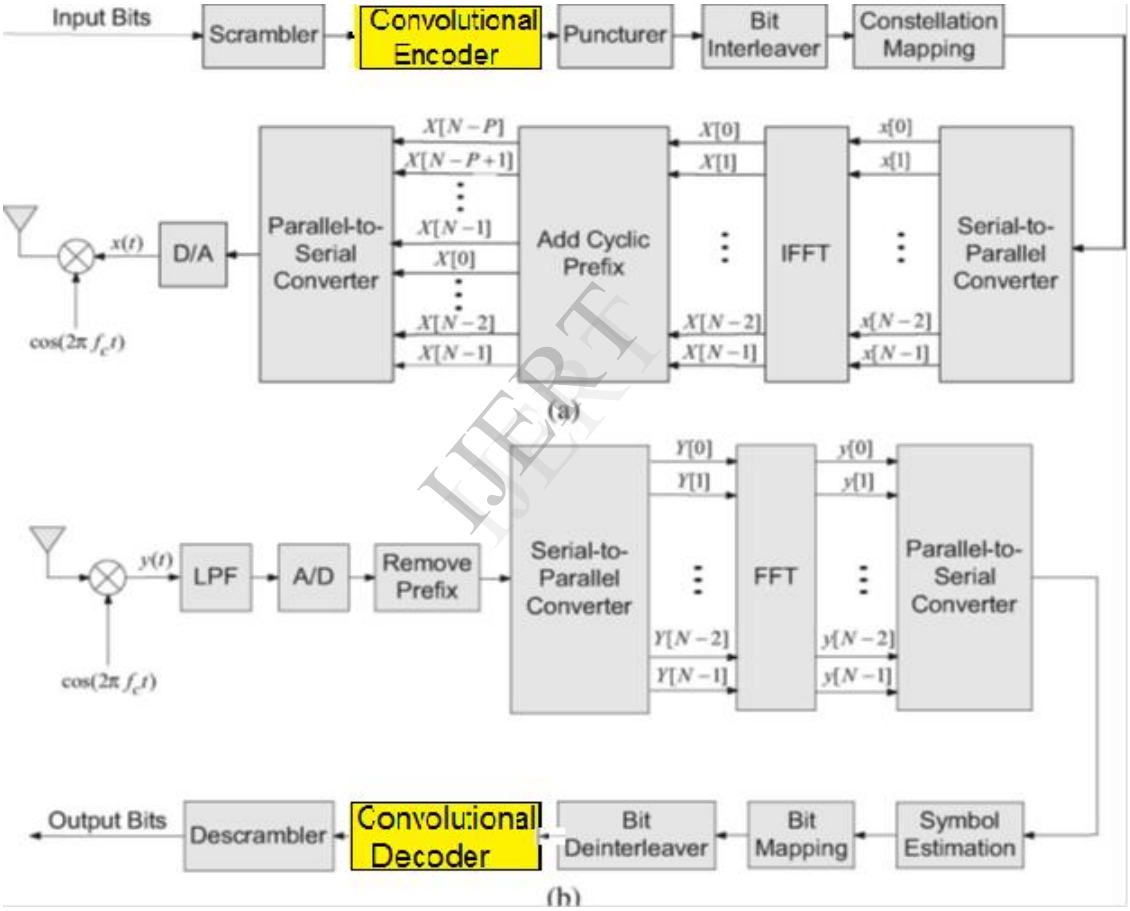

Fig. 1. Block diagrams of the transmitter and receiver of an MB-OFDM system. (a) Transmitter. (b) Receiver.

1.2 ENCODER STRUCTURE

As any binary code, convolutional codes protect informationby adding redundant bits. A rate-*k*/*n* convolutional encoder processes the input sequence of *k*-bit information symbols through one or more binary shift registers (possibly employing feedback). The convolutional encoder computes each *n*-bit symbol (*n* >*k*) of the output sequencefrom linear operations on the current input symbol and the contents of the shift register(s). Thus, a rate *k*/*n*convolutional encoder processes a *k*-bit input symbol and computes an *n*-bit output symbol with every shift register update. Figures 2 and 3 illustrate feedforward and feedback encoder implementations of a rate-1/2 code.explores the similarities and differences between feedforwardand feedback encoders by examining their state diagrams.

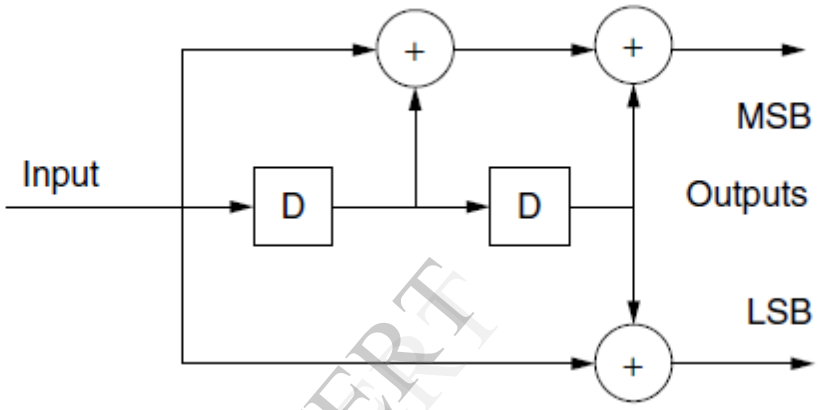

Fig2. Rate-1/2 feedforward convolutional encoder with two memory elements (four states). MSB and LSB refer to the most and least significant bits, respectively.

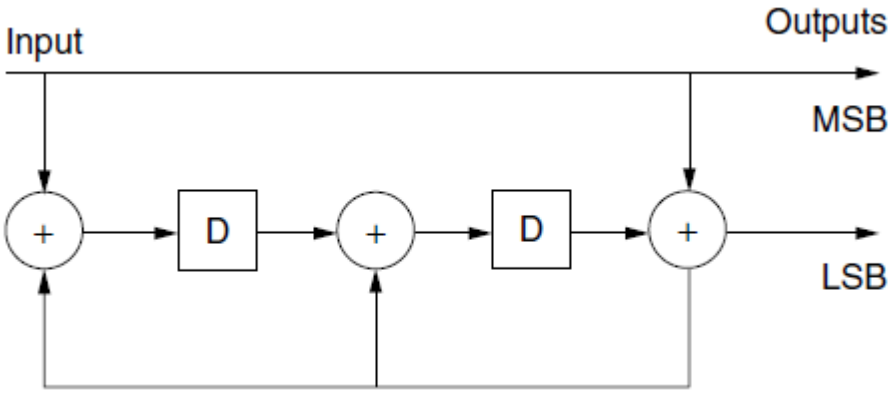

Fig3. Rate-1/2 feedback convolutional encoder with twomemory elements (four states).

1.3 Equivalent Encoders

Convolutional encoders are finite-state machines. Hence, state diagrams provide considerable insight into the behavior of convolutional codes. Figures 4 and 5 provide the state diagrams for the encoders of Figs. 2 and 3, respectively. The states are labeled so that the least significant bit is the one residing in the leftmost memory element of the shift register. The branches are labeled with the 1-bit (single-bit) input and the 2-bit output separated by a comma. The most significant bit (MSB) of the two-bit output is the bit labeled MSB in Figs. 2 and 3. If one erases the state labels and the single-bit input labels, the remaining diagrams for Figs. 4 and 5 (labeled with only the 2-bit outputs) would be identical. This illustrates that the two encoders are equivalent in the sense that both encoders produce the same set of possible output sequences (or codewords). Strictly speaking, a code refers to the list of possible output sequences without specifying the mapping of inputs sequences tooutput sequences. Thus, as in the above example, two equivalent encoders have the same set of possible output sequences, but may implement different mappings from input sequences to output sequences. In the standard convolutional coding application of transmission over additive white Gaussian noise (AWGN) with Viterbidecoding, such encoders give similar BER performance, but the different mappings of inputs to outputs do lead to small performance differences.

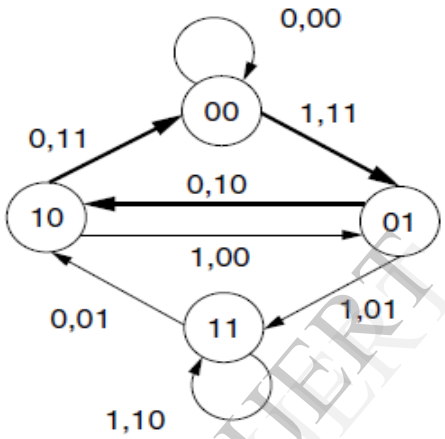

Fig 4.State diagram for rate-1/2 feedforward convolutionalEncoder of Fig. 2.

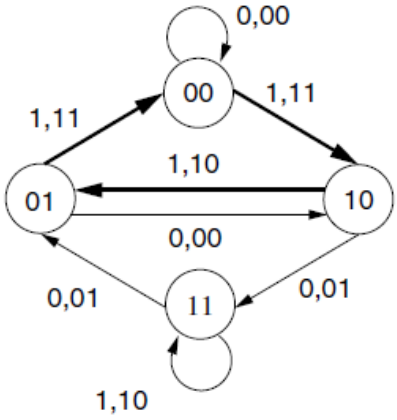

Figure 5.State diagram for rate-1/2 feedback convolutionalencoder of Fig. 3.

The three-branch paths emphasized with thicker arrows in Figs. 4 and 5 are each the shortest nontrivial (i.e., excluding the all-zeros self-loop) loop from the allzeros state back to itself. Notice that for Fig. 4, the statediagram corresponding the feedforward encoder, this loop requires only a single nonzero input. In contrast, for the state diagram corresponding to Fig. 5, this loop requires three nonzero inputs. In fact, for Fig. 5 no nontrivial loop from the all-zeros state to itself requires fewer than twononzero inputs. Thus the feedforward shift register has a finite impulse response, and the feedback shift register has an infinite impulse response. This difference is not particularly important for convolutional codes decoded with Viterbi, but it is extremely important to convolutional encoders used as constituents in Turbo codes, whichare constructed by concatenating convolutional codes separated by interleaves. Only feedback encoders (with infinite impulse responses) are effective constituents in Turbo codes. Thus, equivalent encoders can produce dramatically different performance as constituents in Turbo codes, depending on whether or not they meet the requirement for an infinite impulse response.

1.4 Convolutional encoder:-

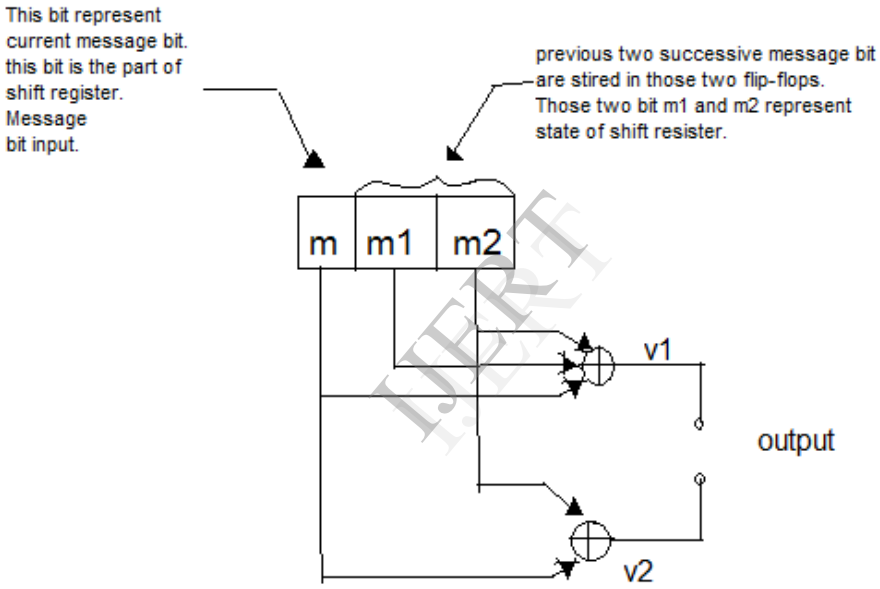

Fig6. convolutional encoder with K=3,k=1 and n=2

Whenever the message bit is shifted to position 'm' the new values of v1 and v2 are generated depending upon m,m1 and m2. M1 and m2 store the previous two message bits. The current bit is present in m. thus we can write

$$V1=m \oplus m1 \oplus m2$$

And $V2=m \oplus m2$

The output switch first samples V1 and then V2. The shift register then shifts contents of m1 to m2 and contents of m to m1. Next input bit is then taken and stored in m again V1and V2 are generated according to this new combination of m,m1 and m2(from V1 and V2). The output switch then samples V1 and V2. Thus the output bit stream for successive input bit will be,

X=V1V2 V1V2 V1V2…….. and so on

Here note that for every input message bit two encoded output bits V1 and V2 are transmitted. In other words, for a single message bit, the encoded code word is two bits i.e. for this convolutional encoder,

Number of message bits ,k=1.

Number of encoded output bit for one message bit, n=2.

Therefore the code rate of this encoder is,

$$r = k/n = 1/2$$

in the encoder observe that whenever a particular message bit enters a shift register, it remains in the shift register for 3 shifts i.e.,

first shift ⟶ message bit is entered in position 'm'.

second shift ⟶ message bit is entered in position 'm1'.

Third shift ⟶ message bit is entered in position 'm2'.

And at the fourth shift the message bit is discarded or simply lost by overwriting. We know that V1 and V2 are combinations of m,m1,m2. Since a single message bit remains in m during first shift, in m1 during second shift and in m2 during third shift. It influences output V1 and V2 for three successive shifts.

1.4.1 States of encoder:

M2	M1	State of encoder
0	0	a
0	1	b
1	0	c
1	1	d

Fig7. States of above convolutional encoder

We have shown that previous two successive message bits m1 and m2 represent state. The input message bit m affects the 'state' of the encoder as well as outputs V1 and V2 during that state. Whenever new message bit is shifted to 'm', the contents of m1 and m2 define new state and output V1 and V2 are also changed according to new state m1,m2 and message bit m. this state shown in above table.

1.4.2 Code tree:-

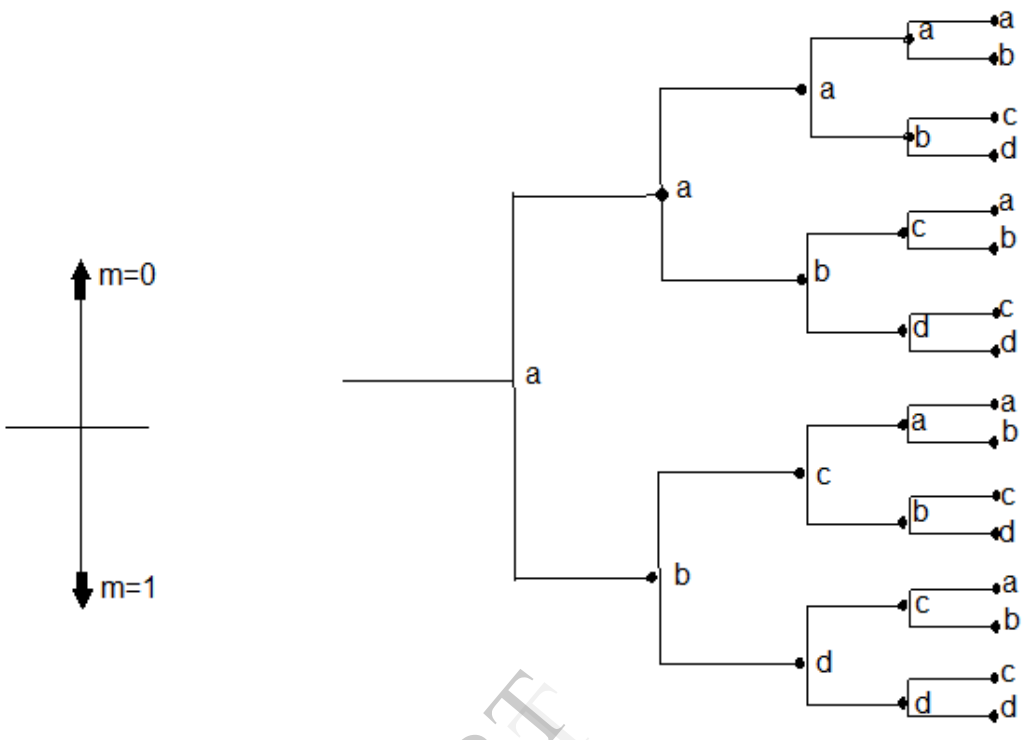

Fig8. Code tree of above convolutional encoder

Thefig8, shows the code tree starts at node 'a'.if input message bit is '1' then path of the tree goes down towards node 'b' and output is formed. Otherwise if input is '0' then path of tree goes up towards node 'a'and output formed.

Example 1. Figure 9 shows a simple rate-1/2 binary linear convolutional encoder. At each time k, one input bit uk comes in, and two output bits (y1k,y2k) go out. The input bits enter a 2-bit shift register, which has 4 possible states (uk−1,uk−2). The output bits are binary linear combinations of the input bit and the stored bits.

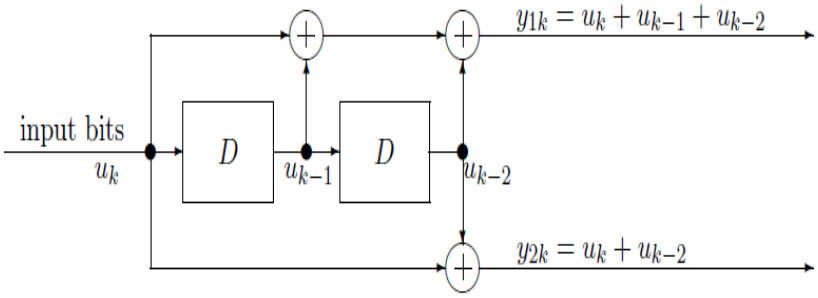

Fig 9.Four-state rate-1/2 binary linear convolutional encoder.

2.1 DECODING CONVOLUTIONAL CODES

Convolutional code decoding algorithms infer the values of the input information sequence from the stream of received distorted output symbols. There are three major families of decoding algorithms for convolutional codes: sequential, Viterbi, and maximum a posteriori (MAP). introducing the trellis structure and showing that Viterbi decoding is maximum-likelihood in the sense that it selects the sequence that makes the received sequence most likely. In 1974, Bahletal.proposed MAP decoding, which explicitly minimizes bit (rather than sequence) error rate. Compared with Viterbi, MAP provides a negligibly smaller bit error rate (and a negligibly larger sequence error rate). These small performance differences require roughly twice the complexity of Viterbi, making MAP unattractive for practical decoding of convolutional codes. However, MAP decoding is crucial to the decoding.

When convolutional codes are used in the traditional way (not as constituents in Turbo codes), they are almost always decoded using some form of the Viterbi algorithm, and the rest of this section focuses on describing it. The goal of the Viterbi algorithm is to find the transmitted sequence (or codeword) that is closest to the received sequence. As long as the distortion is not too severe, this will be the correct sequence.

2.2 Trellis Diagrams

The state diagrams of Figs. 4 and 5 illustrate what transitions are possible from a particular state regardless of time. In contrast, trellis diagrams use a different branch for each different symbol time. As a result, trellis diagrams more clearly illustrates long trajectories throughthe states. Figure 5 shows one stage (one symbol time) of the trellis diagram associated with the rate-1/2 feedforward encoder of Figs. 2 and 4. Each column of states in the trellis diagram includes everything in the original state diagram. All the branches emanating from states in a particular column are incident on the states in the adjacent column

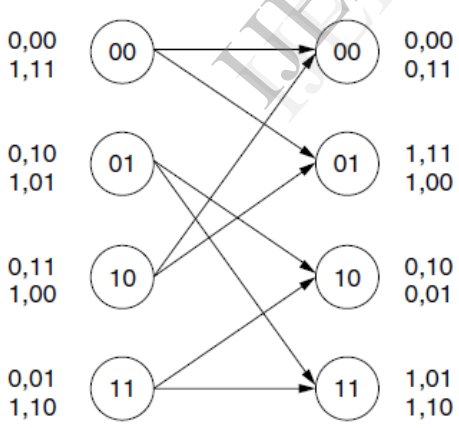

Figure 10.One stage of the trellis diagram for rate-1/2 feedforward convolutional encoder of Figs. 2 and 4.

to the right. In other words, each state transition in the trellis moves the trajectory one stage to the right. To avoid crowding in Fig. 10, branch labels appear at the left and right of the trellis rather than on the branchitself. For each state, the top label belongs to the topbranch emanating from or incident to that state. Figure 11 uses thick arrows to show the same path emphasized in the state diagram of Fig. 3. However, in Fig. 3 the beginning and end of the path were not clear. In Fig. 10 the pathclearly begins in state 00 and then travels through 01 and then 10 before returning to 00.

2.3 The Viterbi Algorithm...

The Viterbi Algorithm (named after Andrew Viterbi) is a dynamic algorithm that uses certain path metrics to compute the 'most likely' path of a transmitted sequence. From this 'most likely' path, certain bit errors can be corrected to decipher the original bit sequence after it has been sent down a communicative line. An important feature of the Viterbi algorithm is that ties are arbitrarily solved (can be picked randomly) and still yield an original sequence. What the Viterbi algorithm can do is correctly replicate your input string at the output even in the presence of one or more errors. Obviously, with more errors introduced the likelihood of a successful decryption does go down. But the algorithm has proved to be effective.

2.3.1The Basic Viterbi Algorithm

The Viterbi algorithm uses the trellis diagram to compute the accumulated distances (called the *path metrics*) from the received sequence to the possible transmitted sequences. The total number of such trellis paths grows exponentially with the number of stages in the trellis, causing potential complexity and memory problems. However, the Viterbi algorithm takes advantage of the fact that the number of paths truly in contention to have the minimum distance is limited to the number of states in a single column of the trellis, assuming that ties maybe arbitrarily resolved.As an example of the Viterbi algorithm, consider transmission over the binary symmetric channel (bit error channel) where the probability of a bit error is less than 1/2.On such a channel, maximum likelihood decoding reduces to finding the output sequence that differs in the fewest bit positions (has the minimum Hamming distance) from the received sequence. For this example, assume the encoder of Fig. 2 with the state diagram of Fig. 4 and the trellis of Fig. 10. For simplicity, assume that the receiver knowsthat the encoder begins in state 00.

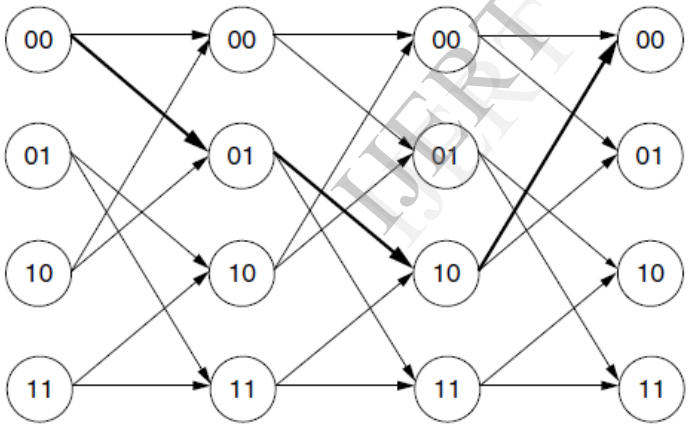

Fig. 11.Trellis diagram for the path emphasized in Fig. 4.

Fig 11 illustrates the basic Viterbi algorithm for the received sequence is 01 01 10. Beginning at the far left column, the only active state is 00. The circle representing this state contains a path metric of zero, indicating that as yet, the received sequence differs from the possible output sequences in no bit positions. Follow the two branches leaving the first column to the active states in the second column of the trellis. Branch metrics label each branch, indicating the Hamming distance between the received symbol and the symbol transmitted by traversing that branch in the encoder. The two hypothetical transmitted symbols are 00 for the top branch and 11 for the bottom (see Fig. 10). Sinceboth differ in exactly one bit position from the received symbol 01, both branch labels are one. The path metric for each destination state is the sum of the branch metric for the incident branch and the path metric at the root of the incident branch. In the second column, both path metrics are one since the root path metric is zero. These equal path metrics indicate that no path is favored at this point. Now follow the branchesfrom the second column to the third. Exactly one branch reaches each state in the third column. Once again, adding the branch metric and the associated root path metric produces the new path metric.

When following branches from the third column to the fourth, two branches are incident on each state. Only the path with the minimum path metric needs to survive. For example, state 00 (the top state) in the fourth column has a path incident from state 00 in the third column with a path metric of 2+ 1 = 3. It also has a path incident from state 10 in the third column with a path metric of 3 + 1 = 4. Only the path with the smaller path metric needs to survive. Figure 12 shows the incident branches of survivor paths with thicker arrows than nonsurvivor paths. Each state in the fourth column has exactly one survivor path, and the values shown indicate the path metrics of the survivor paths.

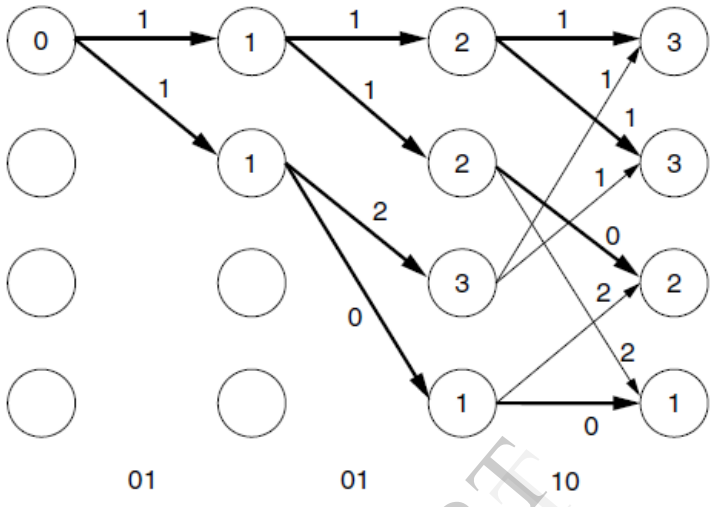

Figure 12.Illustration of the basic Viterbi algorithm on the bit error channel. This is also the trellis for hard Viterbi decoding on the AWGN channel for the case of soft Viterbi decoding shown in Fig. 11.

After all received symbols have been processed, the final step in decoding is to examine the last column and find the state containing the smallest path metric. This is state 11, the bottom state, in the fourth column. Following the survivor branches backward from the minimum-pathmetric state identifies the trellis path of the maximum likelihood sequence. Reference to Fig. 10 reveals that the maximum likelihood path is the state trajectory 00 →01 → 11 →11. This state trajectory produces the output symbol sequence 11 01 10, which differs in exactly one bit position from the received sequence as indicated by its pathmetric. The input information sequence is decoded to be 1 1 1. In this short example, only one trellis stage required path selection. However, once all states are active, path selection occurs with each trellis stage. In fact, if the initial encoder state is not known, path selection occurs even at the very first trellis stage. The basic computational module of the Viterbi algorithm is an add-compare-select (ACS) module. Adding the root path metric and incident branch metric produces the new path metric. Comparing the contending path metrics allows the decoder to select the one with minimum distance. When there is a tie (where two incident paths have the same pathmetric), a surviving path may be arbitrarily selected. In practice, ties are not uncommon. However, ties usually occur between paths that are ultimately destined to be losers.

2.4 Hard versus Soft Decoding

The integer branch and path metrics of the binary error channel facilitate a relatively simple example of the Viterbi algorithm. However, the AWGN channel is far more common than the bit error channel. For the AWGN channel, binary phase shift keying (BPSK) represents binary 1 with 1.0 and binary 0 with −1.0. These twotransmitted values are distorted by additive Gaussiannoise, so that the received values will typically be neither1.0 nor−1.0. A novice might choose to simply quantizeeach received value to the closest of 1.0 and −1.0 andassign the appropriate binary value. This quantizationwould effectively transform the AWGN channel to thebit error channel, facilitating Viterbi decoding exactlyas described above. This method of decoding is calledhard decoding, because the receiver makes a binary (hard)decision about each bit before Viterbi decoding.Hard decoding performs worse by about 2 dB thana more precise form of Viterbi decoding known as *softdecoding*. Soft decoding passes the actual received valuesto the Viterbi decoder. These actual values are calledsoft values because hard decisions (binary decisions) havenot been made to Viterbi decoding. Soft Viterbi decoding isvery similar to hard decoding, but branch and path metricsuse squared Euclidean distance rather than Hammingdistance. works an example analogous to that ofFig. 7 for the case where 1.0 and −1.0 are transmitted overthe AWGN channel and soft Viterbi decoding is employed.A fixed-point implementation with only a few bits ofprecision captures almost all the benefit of soft decoding.

CONCLUSION:-

performance of convolutionalcodes is actually quite good, given their short blocklengths.Applications such as speech transmission that require very low latency continue to employ convolutional codes because they provide excellent performance for their low latency and may be decoded with relatively low complexity. Convolutionalencoders as constituents, a good understanding of convolutional codes remains essential even for long-blocklength applications.Convolutional encoding can be used to improve the performance of wireless systems.Viterbi algorithm is an optimum decoder. Using convolutional coding, the information can beextracted without any error from noisy channel.

FUTURE SCOPE

Work presented in my Paper can be extended to evaluate the performance of convolutional coding for orthogonal frequency division multiplexing (OFDM) system and multicarrier code division multiple access (MC-CDMA) system. Our results can also be extended to include the performance of DS-CDMA system with different modulation schemes using convolutional coding over Rayleigh, Nakagami and generalized fading channel.

REFERANCES:-

1. P. Elias, Coding for noisy channels, *Proc. IRE Conv. Rec.part 4*• 37–46 (1955) (this paper is also available in Ref. 2). Q2
2. E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, IEEE Press, 1974.
3. S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.
4. R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, 1999.
5. G. D. Forney, Jr., Convolutional codes I: Algebraic structure, *IEEE Trans. Inform. Theory* **16**(6): 720–738 (Nov. 1970).
6. R. Johannesson and Z. Wan, A linear algebra approach to minimal convolutional encoders, *IEEE Trans. Inform. Theory***39**(4): 1219–1233 (July 1993).