CHANNEL CODING THEOREM

For a discrete memoryless channel, represents the maximum amount of information that can be transmitted per channel use in a reliable manner.

The *channel coding theorem* states:

If a discrete memoryless channel has capacity *C* and a source generates information at a rate less than *C*, then there exists a coding technique such that the output of the source may be transmitted over the channel with an arbitrarily low probability of symbol error.

For the special case of a binary symmetric channel, the theorem teaches us that if the code rate *r* is less than the channel capacity *C*, then it is possible to find a code that achieves error-free transmission over the channel. Conversely, it is not possible to find such a code if the code rate *r* is greater than the channel capacity *C*. Thus, the channel coding theorem specifies the channel capacity *C* as a *fundamental limit* on the rate at which the transmission of reliable (error-free) messages can take place over a discrete memoryless channel. The issue that matters here is not the SNR, so long as it is large enough, but how the channel input is encoded.

The most unsatisfactory feature of the channel coding theorem, however, is its nonconstructive nature. The theorem asserts the existence of good codes but does not tell us how to find them. By *good codes* we mean families of channel codes that are capable of providing reliable transmission of information (i.e., at arbitrarily small probability of symbol error) over a noisy channel of interest at bit rates up to a maximum value less than the capacity of that channel. The error-control coding techniques described in this chapter provide different methods of designing good codes

Notation

Many of the codes described in this chapter are *binary codes*, for which the alphabet consists only of binary symbols 0 and 1. In such a code, the encoding and decoding functions involve the binary arithmetic operations of *modulo-2 addition and multiplication* performed on codewords in the code.

Throughout this chapter, we use the ordinary plus sign (+) to denote modulo-2 addition. The use of this terminology will not lead to confusion because the whole chapter relies on binary arithmetic.

Thus, according to the notation used in this chapter, the rules for modulo-2 addition are as follows: Because 1 + 1 = 0, it follows that 1 = –1. Hence, in binary arithmetic, subtraction is the same as addition. The rules for modulo-2 multiplication are as follows:

Division is trivial, in that we have and division by 0 is not permitted. Modulo-2 addition is the EXCLUSIVE-OR operation in logic and modulo-2 multiplication is the AND operation.

Linear Block Codes

By definition:

A code is said to be linear if any two codewords in the code can be added in modulo-2 arithmetic to produce a third codeword in the code.

Consider, then, an (n,k) linear block code, in which k bits of the n code bits are always Identical to the message sequence to be transmitted. The ($n – k$) bits in the remaining portion are computed from the message bits in accordance with a prescribed encoding rule that determines the mathematical structure of the code. Accordingly, these ($n – k$) bits are referred to as *parity-check bits*. Block codes in which the message bits are transmitted in unaltered form are called *systematic codes*. For applications requiring *both* error detection and error correction, the use of systematic block codes simplifies implementation of the decoder.

In the linear block codes, the parity bits and message bits have a linear combination, which means that the resultant code word is the linear combination of any two code words.

Let us consider some blocks of data, which contains **k** bits in each block. These bits are mapped with the blocks which has **n** bits in each block. Here **n** is greater than **k**. The transmitter adds redundant bits which are n−kn−k bits. The ratio **k/n** is the **code rate**. It is denoted by **r** and the value of **r** is **r < 1**.

The n−kn−k bits added here, are **parity bits**. Parity bits help in error detection and error correction, and also in locating the data. In the data being transmitted, the left most bits of the code word correspond to the message bits, and the right most bits of the code word correspond to the parity bits.

Block codes operate on a block of bits. Block codes are referred to as (n, k) codes. A block of k information bits are coded to become a block of n bits. But before we go any further with the details, lets look at an important concept in coding called Hamming distance. Lets say that we want to code the 10 integers, 0 to 9 by a digital sequence. Sixteen unique sequences can be obtained from four bit words. We assign the first ten of these, one to each integer. Each integer is now identified by its own unique sequence of bits.

Let $m0$, $m1$, , mk – 1 constitute a block of k arbitrary message bits. Thus, we have $2k$ distinct message blocks. Let this sequence of message bits be applied to a linear block

$0 + 0 = 0$

$1 + 0 = 1$

$0 + 1 = 1$

$1 + 1 = 0$

Because $1 + 1 = 0$, it follows that $1 = –1$. Hence, in binary arithmetic, subtraction is the same as addition. The rules for modulo-2 multiplication are as follows:

Division is trivial, in that we have

and division by 0 is not permitted. Modulo-2 addition is the EXCLUSIVE-OR operation in logic and modulo-2 multiplication is the AND operation.

Creating block codes: The block codes are specified by (n.k). The code takes k information bits and computes (n-k) parity bits from the code generator matrix. Most block codes are systematic in that the information bits remain unchanged with parity bits attached either to the front or to the back of the information sequence.

* Hamming code, a simple linear block code
* Hamming codes are most widely used linear block codes.
* A Hamming code is generally specified as (2n- 1, 2n-n-1).
* The size of the block is equal to 2n-1.

The number of information bits in the block is equal to 2n-n-1 and the number of overhead bits is equal to n. All Hamming codes are able to detect three errors and correct one.

Cyclic Codes

The cyclic property of code words is that any cyclic-shift of a code word is also a code word. Cyclic codes follow this cyclic property.

For a linear code **C**, if every code word i.e., **C =** C1,C2,......CnC1,C2,......Cn from C has a cyclic right shift of components, it becomes a code word. This shift of right is equal to **n-1** cyclic left shifts. Hence, it is invariant under any shift. So, the linear code **C**, as it is invariant under any shift, can be called as a **Cyclic code**.

Cyclic codes are used for error correction. They are mainly used to correct double errors and burst errors.

Hence, these are a few error correcting codes, which are to be detected at the receiver. These codes prevent the errors from getting introduced and disturb the communication. They also prevent the signal from getting tapped by unwanted receivers.

Basic Definition and Examples

Definition

A code C is cyclic if (i) C is a linear code; (ii) any cyclic shift of a codeword is also a codeword, i.e. whenever a0, . . . an−1 ∈ C, then also an−1a0 . . . an–2 ∈ C and a1a2 . . . an−1a0 ∈ C.

Example (i) Code C = {000, 101, 011, 110} is cyclic.

(ii) Hamming code Ham(3, 2): with the generator matrix G = 2 6 6 4 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 1 1 1 3 7 7 5 is equivalent to a cyclic code.

(iii) The binary linear code {0000, 1001, 0110, 1111} is not cyclic, but it is equivalent to a cyclic code.

(iv) Is Hamming code Ham(2, 3) with the generator matrix » 1 0 1 1 0 1 1 2

Cyclic codes form a subclass of linear block codes. Indeed, many of the important linear block codes discovered to date are either cyclic codes or closely related to cyclic codes. An advantage of cyclic codes over most other types of codes is that they are easy to encode. Furthermore, cyclic codes possess a well-defined mathematical structure, which has led to the development of very efficient decoding schemes for them. A binary code is said to be a *cyclic code* if it exhibits two fundamental properties:

PROPERTY 1 **Linearity Property**

The sum of any two codewords in the code is also a codeword.

PROPERTY 2 **Cyclic Property**

Any cyclic shift of a codeword in the code is also a codeword.

Property 1 restates the fact that a cyclic code is a linear block code (i.e., it can be described as a parity-check code). To restate Property 2 in mathematical terms, let the *n*-tuple denote a codeword of an linear block code. The code is a cyclic code if the *n*-tuples

$cn – 2$ $cn – 1 + cn – 3$

$c1$ $c2 + cn – 1$ $c0$

are all codewords in the code.

To develop the algebraic properties of cyclic codes, we use the elements of a codeword **c**+X+ $c0$ $c1X$ $c2X2$ + cn 1 – Xn – 1to define the code polynomial where X is an indeterminate. Naturally, for binary codes, the coefficients are 1s and 0s. Each power of X in the polynomial represents a one-bit *shift* in time. Hence, multiplication of the polynomial by X may be viewed as a shift to the right. The key question is: How do we make such a shift *cyclic*? The answer to this question is addressed next.

Let the code polynomial in (10.27) be multiplied by Xi, yielding

C+X+ $c0$ $c1X$ $c2X2$ + cn 1 – Xn – 1

where X is an indeterminate. Naturally, for binary codes, the coefficients are 1s and 0s. Each power of X in the polynomial represents a one-bit *shift* in time. Hence, multiplication of the polynomial by X may be viewed as a shift to the right If **c**(X) is a code polynomial, then the polynomial is also a code polynomial for any cyclic shift i; the term mod is the abbreviation for modulo

Reed-Solomon Codes:

Reed Solomon (R-S) codes form an important sub- class of the family of Bose-Chaudhuri-Hocquenghem (BCH) codes and are very powerful linear non-binary block codes capable of correcting multiple random as well as burst errors. They have an important feature that the generator polynomial and the code symbols are derived from the same finite field. This enables to reduce the complexity and also the number of computations involved in their implementation. A large number of R-S codes are available with different code rates.

An R-S code is described by a generator polynomial g(x) and other usual important code parameters such as the number of message symbols per block (k), number of code symbols per block (n), maximum number of erroneous symbols (t) that can surely be corrected per block of received symbols and the designed minimum symbol Hamming distance (d). A parity-check polynomial h

(X) of order k also plays a role in designing the code. The symbol x, used in polynomials is an indeterminate which usually implies unit amount of delay.

For positive integers m and t, a primitive (n, k, t) R-S code is defined as below: Number of encoded symbols per block: n = 2m – 1 Number of message symbols per block: k Code rate: R= k/n Number of parity symbols per block: n

– k = 2t Minimum symbol Hamming distance per block: d = 2t +1. It can be noted that the block length n of an (n, k, t) R-S code is bounded by the corresponding finite field GF(2m). Moreover, as n – k = 2t, an (n, k, t) R-S code has optimum error correcting capability.

APPLICATIONS of REED-SOLOMON CODES

- Reed-Solomon codes have been widely used in mass storage systems to correct the burst errors caused by media defects.
- Special types of Reed-Solomon codes have been used to overcome unreliable nature of data transmission over erasure channels.
- Several bar-code systems use Reed-Solomon codes to allow correct reading even if a portion of a bar code is damaged.
- Reed-Solomon codes were used to encode pictures sent by the Voyager spacecraft. Modern versions of concatenated Reed-Solomon/Viterbi decoder convolution coding were and are used on the Mars Pathfinder, Galileo, Mars exploration Rover and Cassini missions, where they performed within about 1-1.5dB of the ultimate limit imposed by the shannon capacity

HAMMING CODES:

The linearity property of the code word is that the sum of two code words is also a code word. Hamming codes are the type of **linear error correcting** codes, which can detect up to two bit errors or they can correct one bit errors without the detection of uncorrected errors.

While using the hamming codes, extra parity bits are used to identify a single bit error. To get from one-bit pattern to the other, few bits are to be changed in the data. Such number of bits can be termed as **Hamming distance**. If the parity has a distance of 2, one-bit flip can be detected. But this can't be corrected. Also, any two bit flips cannot be detected.

However, Hamming code is a better procedure than the previously discussed ones in error detection and correction.

Hamming Weight: The Hamming weight of this code scheme is the largest number of 1‟s ina valid code word. This number is 3 among the 10 code words we have chosen. (the ones in the while space).

Concept of Hamming Distance: In continuous variables, we measure distance by Euclidean concepts such as lengths, angles and vectors. In the binary world, distances are measured between two binary words by something called the Hamming distance. The Hamming distance is the number of disagreements between two binary sequences of the same size. The Hamming distance between sequences 001 and 101 is = 1 The Hamming distance between sequences 0011001 and 1010100 is = 4. Hamming ***distance*** and ***weight*** are very important and useful concepts in coding. The knowledge of Hamming distance is used to determine the capability of a code to detect and correct errors. Although the Hamming ***weight*** of our chosen code set is 3, the minimum Hamming ***distance*** is 1. We can generalize this to say that the maximum number of error bits that can be detected is t = dmin –1.

Where dmin is Hamming distance of the code words.

For a code with dmin = 3, we can both detect 1 and 2 bit errors. So we want to have a code set with as large a Hamming distance as possible since this directly effects our ability to detect errors. The number of errors that we can correct is given by

$$t(\text{int}) = \frac{d_{\min} - 1}{2}$$

For any positive integer *m* > 3, there exists a linear block code with the following parameters:

code length $n = 2m - 1$

number of message bits $k = 2m - m - 1$ number of

parity-check bits $n - k = m$

Such a linear block code for which the error-correcting capability $t = 1$ is called a Hamming code.3 To be specific, consider the example of $m = 3$, yielding the(7, 4)

Hamming code with $n = 7$ and $k = 4$. The generator of this code is defined by

$$\mathbf{G} = \left[\begin{array}{ccc:cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\underbrace{\qquad\quad}_{\mathbf{P}} \qquad \underbrace{\qquad\qquad}_{\mathbf{I}_k}$$

$$\mathbf{H} = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

$$\underbrace{\quad\quad}_{\mathbf{I}_{n-k}} \quad \underbrace{\quad\quad}_{\mathbf{P}^{\mathrm{T}}}$$

The operative property embodied in this equation is that the columns of the parity-check matrix **P** consist of all the nonzero *m*-tuples, where *m* = 3. With *k* = 4, there are 2*k* = 16 distinct message words, which are listed in Table 10.1. For ma given message word, the corresponding codeword is obtained by using (10.13). Thus, the application of this equation results in the 16 codewords listed in Table 10.1.

In Table 10.1, we have also listed the Hamming weights of the individual codewords in the (7,4) Hamming code. Since the smallest of the Hamming weights for the nonzero codewords is 3, it follows that the minimum distance of the code is 3, which is what it should be by definition. Indeed, all Hamming codes have the property that the minimum distance *d*min = 3, independent of the value assigned to the number of parity bits *m*. To illustrate the relation between the minimum distance *d*min and the structure of the parity-check matrix **H**, consider the codeword 0110100.

Table 10.1 Codewords of a (7,4) Hamming code

Message word	Codeword	Weight of codeword	Message word	Codeword	Weight of codeword
0000	0000000	0	1000	1101000	3
0001	1010001	3	1001	0111001	4
0010	1110010	4	1010	0011010	3
0011	0100011	3	1011	1001011	3
0100	0110100	3	1100	1011100	4
0101	1100101	4	1101	0001101	3
0110	1000110	3	1110	0101110	4
0111	0010111	4	1111	1111111	7

An important property of binary Hamming codes is that they satisfy the condition of with the equality sign, assuming that $t = 1$. Thus, assuming single-error patterns, we may formulate the error patterns listed in the right-hand column of Table 10.2. The corresponding eight syndromes, listed in the left-hand column, are calculated in accordance with (10.20). The zero syndrome signifies no transmission errors.

Suppose, for example, the code vector [1110010] is sent and the received vector is [1100010] with an error in the third bit. Using (10.19), the syndrome is calculated to be

$$\mathbf{s} = [1100010]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

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

the corresponding coset leader (i.e., error pattern with the highest probability of occurrence) is found to be [0010000], indicating correctly that the third bit of the received vector is erroneous. Thus, adding this error pattern to the received vector, in accordance yields the correct code vector actually sent.

Table 10.2 Decoding table for the (7,4) Hamming code defined in Table 10.1

Syndrome	Error pattern
000	0000000
100	1000000
010	0100000
001	0010000
110	0001000
011	0000100
111	0000010
101	0000001

Encoding a message by Hamming Code

The procedure used by the sender to encode the message encompasses the following steps

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Calculating the values of each redundant bit.

Once the redundant bits are embedded within the message, this is sent to the user.

Step 1 − Calculation of the number of redundant bits.

If the message contains *mm*number of data bits, *rr*number of redundant bits are added to it so that *mr* is able to indicate at least (*m* + *r*+ 1) different states. Here,

$(m + r)$ indicates location of an error in each of $(m + r)$ bit positions and one additional state indicates no error. Since, rr bits can indicate $2^r r$ states, $2^r r$ must be at least equal to $(m + r + 1)$. Thus the following equation should hold $2^r \geq$ m+r+

Step 2 − Positioning the redundant bits.

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc. They are referred in the rest of this text as r_1 (at position 1), r_2 (at position 2), r_3 (at position 4), r_4 (at position 8) and so on.

Step 3 − Calculating the values of each redundant bit.

The redundant bits are parity bits. A parity bit is an extra bit that makes the number of 1s either even or odd. The two types of parity are −

- **Even Parity** − Here the total number of bits in the message is made even.
- **Odd Parity** − Here the total number of bits in the message is made odd.

Each redundant bit, r_i, is calculated as the parity, generally even parity, based upon its bit position. It covers all bit positions whose binary representation includes a 1 in the ith position except the position of r_i. Thus −

- r_1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)
- r_2 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)
- r_3 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)

Decoding a message in Hamming Code

Once the receiver gets an incoming message, it performs recalculations to detect errors and correct them. The steps for recalculation are −

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Parity checking.
- **Step 4** − Error detection and correction

Step 1 − Calculation of the number of redundant bits

Using the same formula as in encoding, the number of redundant bits are ascertained.
$2^r \geq m + r + 1$ where *m* is the number of data bits and *r* is the number of redundant bits.

Step 2 − Positioning the redundant bits

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

Step 3 − Parity checking

Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of c_1,c_2 ,c_3 ,c_4 etc. Thus

c_1 = parity(1, 3, 5, 7, 9, 11 and so on)

c_2 = parity(2, 3, 6, 7, 10, 11 and so on)
c_3 = parity(4-7, 12-15, 20-23 and so on)

Step 4 − Error detection and correction

The decimal equivalent of the parity bits binary values is calculated. If it is 0, there is no error. Otherwise, the decimal value gives the bit position which has error. For example, if $c_1c_2c_3c_4$ = 1001, it implies that the data bit at position 9, decimal equivalent of 1001, has error. The bit is flipped to get the correct message.

Viterbi Decoding Algorithm

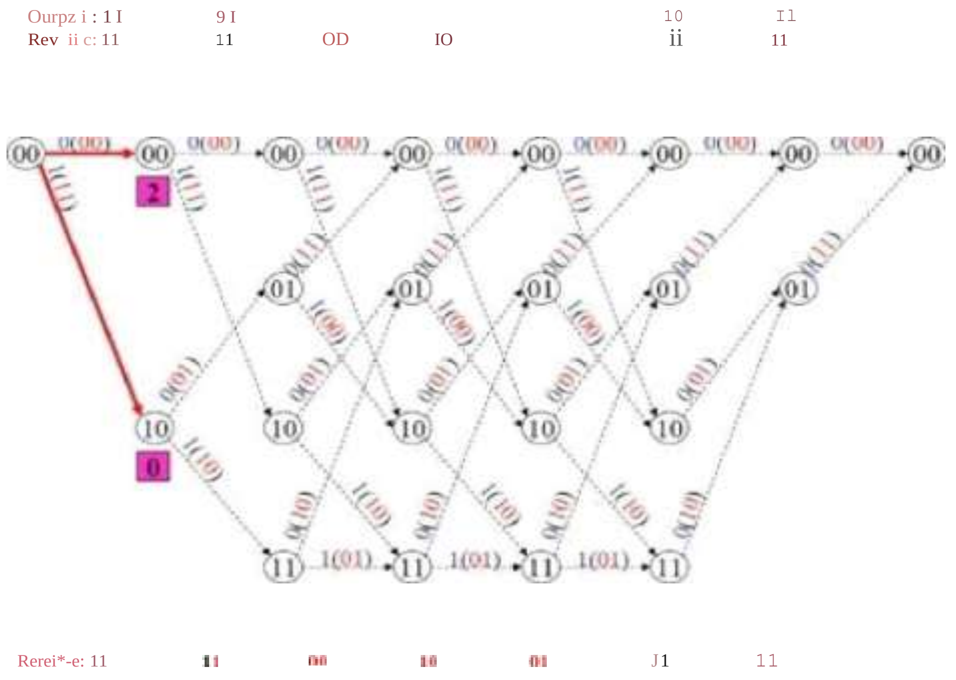
An efficient search algorithm, Performing ML decoding rule.

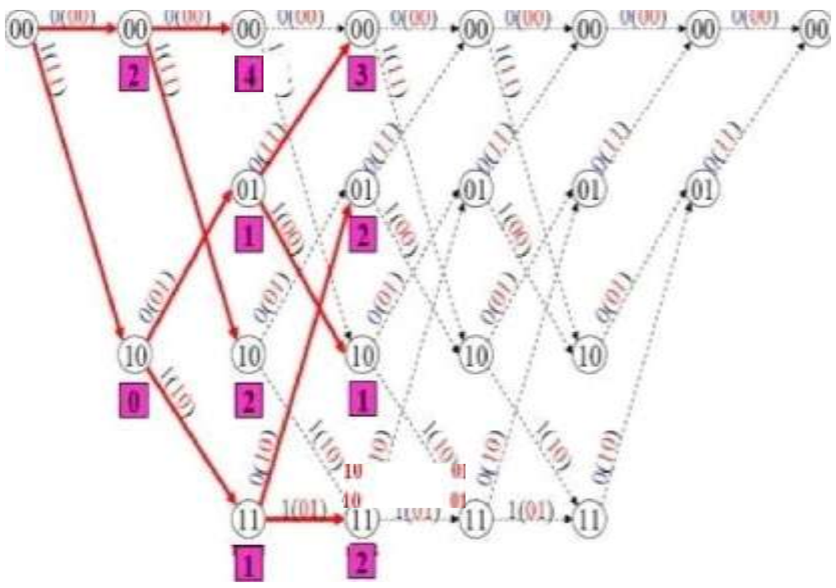
Viterbi Decoding Algorithm

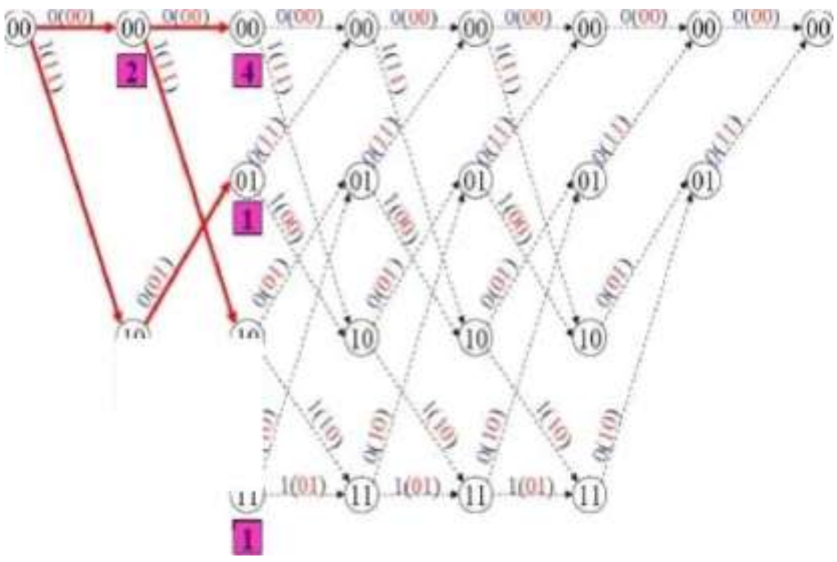
- An efficient search algorithm
- Performing ML decoding rule.
- Reducing the computational complexity.

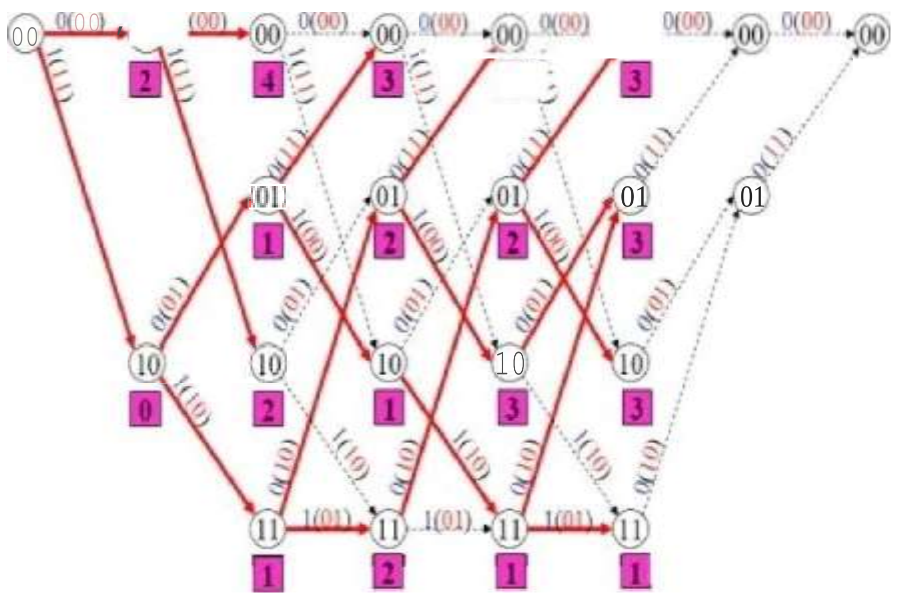
Basic concept

- Generate the code trellis at the decoder
- The decoder penetrates through the code trellis *level by level* in search for the transmitted code sequence
- At each level of the trellis, the decoder computes and compares the metrics of all the partial paths entering a node
- The decoder *stores* the partial path with the larger metric and *eliminates* all the other partial paths. The stored partial path is called the *survivor*.

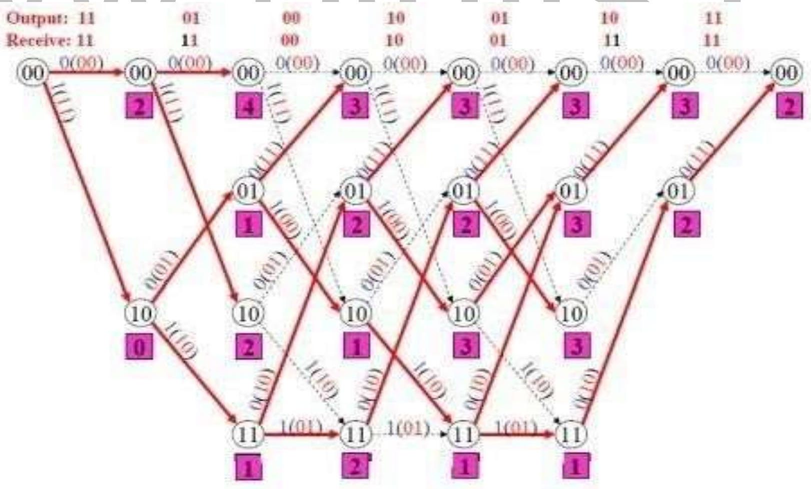




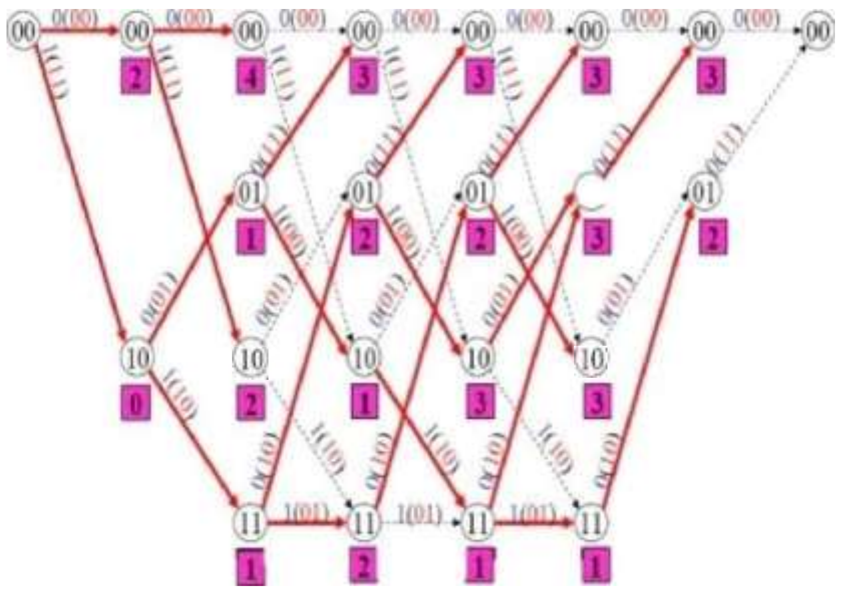


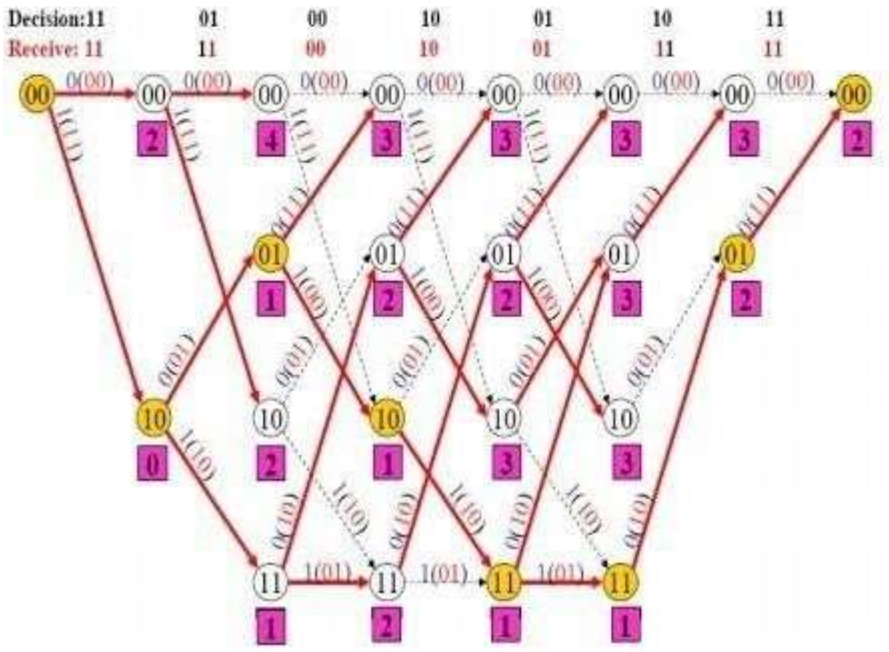


Receive: 11 11 00 10



01





(Source: J.G Proakis, ―Digital Communication‖, 4th Edition, Tata Mc Graw Hill Company, 2001)