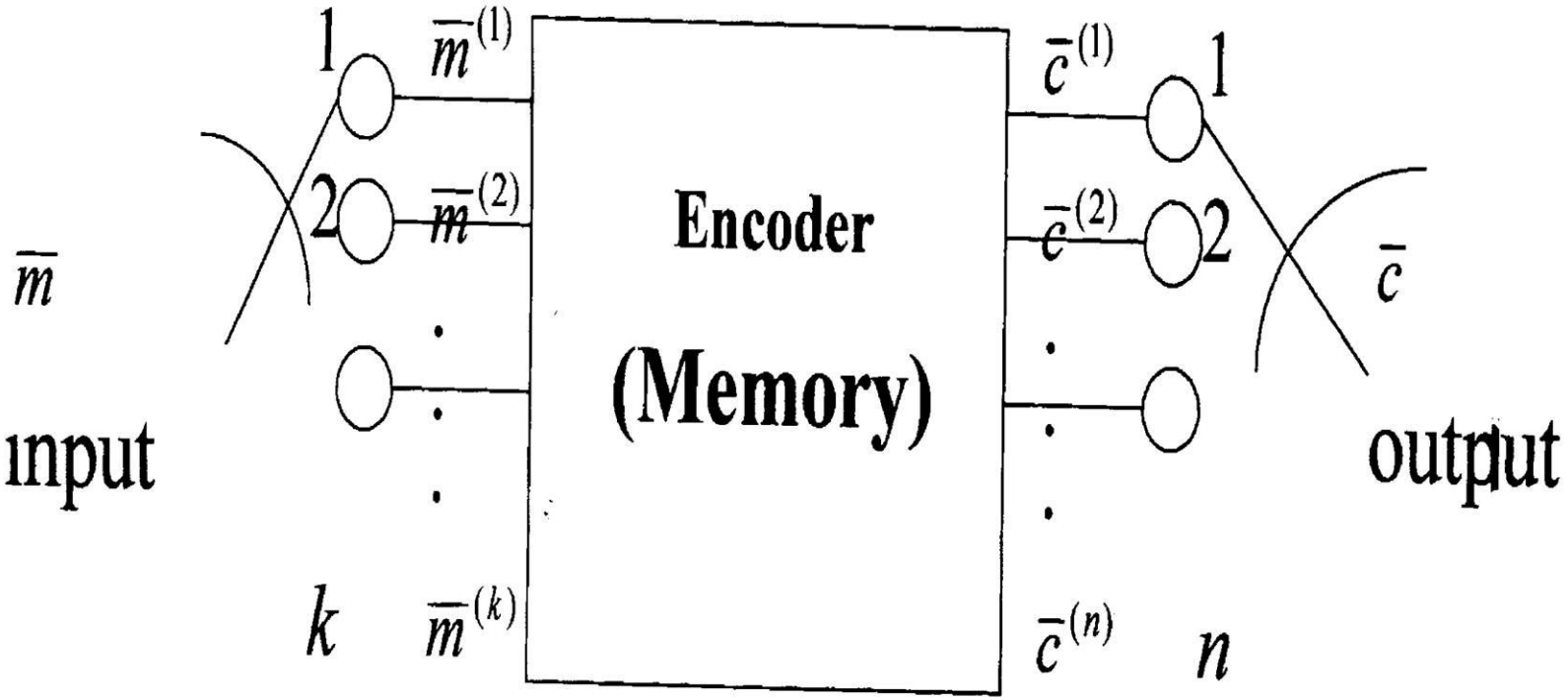
7.2 Convolutional Codes

7.2.1 Introduction

- **Convolutional codes were first discovered by P. Elias in 1955.**
- **The structure of convolutional codes is quite different from that of block codes.**

  During each unit of time, the input to a convolutional code encoder is also a *k*-bit message block and the corresponding output is also an *n*-bit coded block with $k < n$.
- **Each coded *n*-bit output block depends not only the corresponding *k*-bit input message block at the same time unit but also on the *M* previous message blocks.**
- **Thus the encoder has *k* input lines, *n* output lines and a memory of order *M* as shown in Figure 7.1 .**

Fig. 7.1


$\bar{m}$
input
1
2
k
$\bar{m}^{(1)}$
$\bar{m}^{(2)}$
$\bar{m}^{(k)}$
Encoder
(Memory)
$\bar{c}^{(1)}$
$\bar{c}^{(2)}$
$\bar{c}^{(n)}$
1
2
n
$\bar{c}$
output

- **Each message (or information) sequence is encoded into a code sequence.**
- **The set of all possible code sequences produced by the encoder is called an (n,k,M) convolutional code.**

  The parameters, k and n , are normally small,

  say $1 \leq k \leq 8$ and $2 \leq n \leq 9$.

  The ratio $R = k/n$ is called the code rate.

  The parameter M is called the memory order of the code.
- **Note that the number of redundant (or parity) bits in each coded block is small.**

  However, more redundant bits are added by increasing the memory order M of the code while holding k and n fixed.

7.2.2 (*n,k,M*) Convolutional Codes

- For (*n,k,M*) code , the encoder has *k* inputs and *n* outputs as shown in Fig. 8.1
- At the *i*-th input terminal, the input message sequence is

$$\mathbf{m}^{(i)} = (m_1^{(i)}, m_2^{(i)}, \ldots, m_r^{(i)}, \ldots) \qquad \text{for } 1 \le i \le k. \qquad (1)$$

- At the *j*-th output terminal, the output code sequence is

$$\mathbf{c}^{(j)} = (c_1^{(j)}, c_2^{(j)}, \ldots, c_r^{(j)}, \ldots) \qquad \text{for } 1 \le j \le n \qquad (2)$$

- **An (*n*,*k*,*M*) convolutional code is specified by $k \times n$ generator sequence:**

$$\begin{matrix} \overline{g}_1^{(1)}, & \overline{g}_1^{(2)}, & \cdots & \overline{g}_1^{(n)} \\ \overline{g}_2^{(1)} & \overline{g}_2^{(2)} & \cdots & \overline{g}_2^{(n)} \\ \vdots & \vdots & & \vdots \\ \overline{g}_k^{(1)} & \overline{g}_k^{(2)} & \cdots & \overline{g}_k^{(n)} \end{matrix}$$

The *n* output code sequences are then given by

$$\overline{c}^{(1)} = \overline{m}^{(1)} * \overline{g}_1^{(1)} + \overline{m}^{(2)} * \overline{g}_2^{(1)} + \cdots + \overline{m}^{(k)} * \overline{g}_k^{(1)}$$

$$\overline{c}^{(2)} = \overline{m}^{(1)} * \overline{g}_1^{(2)} + \overline{m}^{(2)} * \overline{g}_2^{(2)} + \cdots + \overline{m}^{(k)} * \overline{g}_k^{(2)}$$

$$\vdots$$

$$\overline{c}^{(n)} = \overline{m}^{(1)} * \overline{g}_1^{(n)} + \overline{m}^{(2)} * \overline{g}_2^{(n)} + \cdots + \overline{m}^{(k)} * \overline{g}_k^{(n)}$$

(4)

Encoder

- **The encoder of an (*n*,*k*,*M*) code consists of *k* shift-registers, each has at most *M* stages. The feedforward connections are based on the *k*×*n* generator sequences.**
- **The message bits stored in the *k* shift-registers together represent the state of the encoder.**

- **Example 7.1**

  Let *n*=3, *k*=2, and *M* =1. Consider the (3,2,1) convolutional code generated by the following 6 generator sequences:

$$\bar{g}_1^{(1)} = (1 \quad 1), \quad \bar{g}_1^{(2)} = (0 \quad 1), \quad \bar{g}_1^{(3)} = (1 \quad 1),$$

$$\bar{g}_2^{(1)} = (0 \quad 1), \quad \bar{g}_2^{(2)} = (1 \quad 0), \quad \bar{g}_2^{(3)} = (1 \quad 0).$$

- **The output sequence are:**

$$\bar{c}^{(1)} = \bar{m}^{(1)} * \bar{g}_1^{(1)} + \bar{m}^{(2)} * \bar{g}_2^{(1)} \qquad \textbf{(5)}$$

$$\bar{c}^{(2)} = \bar{m}^{(1)} * \bar{g}_1^{(2)} + \bar{m}^{(2)} * \bar{g}_2^{(2)}$$

$$\bar{c}^{(3)} = \bar{m}^{(1)} * \bar{g}_1^{(3)} + \bar{m}^{(2)} * \bar{g}_2^{(3)}$$

- **The 3 code digits of the *l*-th code block are given by:**

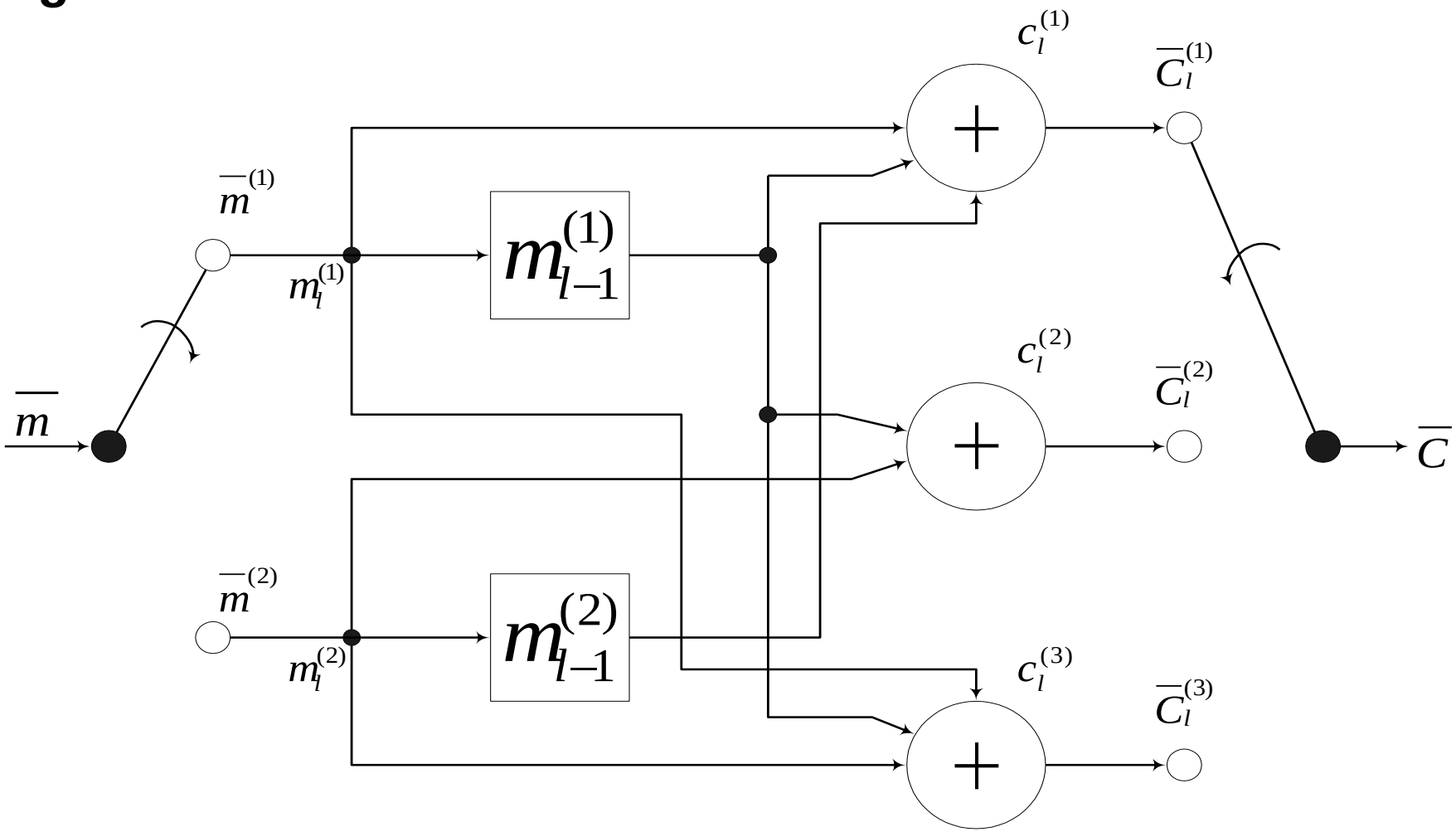
$$c_l^{(1)} = m_l^{(1)} + m_{l-1}^{(1)} + m_{l-1}^{(2)}$$

$$c_l^{(2)} = m_{l-1}^{(1)} + m_l^{(2)} \qquad \textbf{(6)}$$

$$c_l^{(3)} = m_l^{(1)} + m_{l-1}^{(1)} + m_l^{(2)}$$

- **The encoder is shown in Figure 7.2**

Fig.7.2



- **Transform –Domain Operation :**

Generator Polynomial G(D)

$$G(D) = \begin{matrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \dots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \dots & g_2^{(n)}(D) \\ \vdots & \vdots & & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \dots & g_k^{(n)}(D) \end{matrix}$$

The encoder output can be expressed by

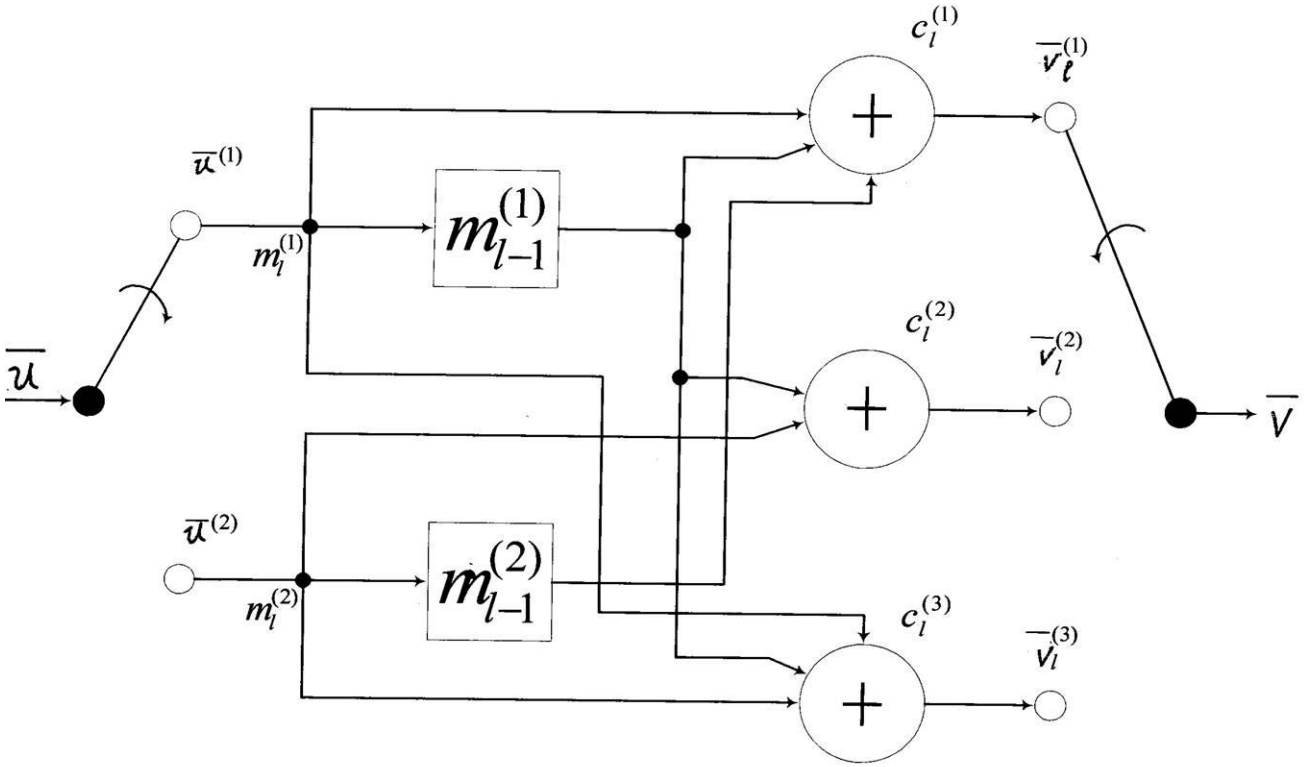
$$V(D) = U(D)\, G(D)$$

where $U(D) = [\, u^{(1)}(D),\ u^{(2)}(D)\ , \dots,\ u^{(k)}(D)\,]$ **…..Input**

$V(D) = [\, v^{(1)}(D),\ v^{(2)}(D)\ , \dots,\ v^{(n)}(D)\,]$ **….Output**

Example : The (3,2,1) encoder shown in Fig.7.2

$$G(D) = \begin{matrix} 1+D & D & 1+D \\ D & 1 & 1 \end{matrix} \qquad V(D) = (U_1 \; U_2) \begin{matrix} 1+D & D & 1+D \\ D & 1 & 1 \end{matrix}$$



7.2.3 Systematic Form

- An (*n, k, M*) convolutional code is said to be systematic if the first output sequence is identical to the input message sequence , i.e.,

$$C^{(1)} = m \qquad (7)$$

Suppose the input message sequence is of finite length with *L* bits,

$$m^{(i)} = (m_1^{(i)}, m_2^{(i)}, \ldots, m_L^{(i)}) \qquad (8)$$

Since it take the last bit, m_{L-1} , *M* units of time to move out of the memory, each output sequence consists of *L+M* bits,

$$c^{(j)} = (v_1^{(j)}, v_2^{(j)}, \ldots, v_j^{(j)}, \ldots, v_{L+M}^{(j)})$$

for $i = 1, 2, \ldots, n$. (9)

- After *M* zeros are added to the message sequence to compute the last *M* output blocks (clear the shift register).
- The parameter $K = M + 1$ is called the constraint length of the code.

7.2.4 State Diagram

- **Since the encoder is a linear sequential circuit, its behavior can be described by a state diagram.**

  Define the encoder state at the time *l* as the *M*-tuple,

  $$(m_{l-1}, m_{l-2}, \ldots, m_{l-M})$$

  which consists of the *M* message bits stored in the shift registers.

- **There are 2^M possible states. At any time instant, the encoder must be in one of these states.**
- **The encoder undergoes a state transition when a message bit is shifted into the encoder register as shown below.**

Input	State
m_l	$(m_{l-1}, m_{l-2}, \ldots, m_{l-M})$
	$\leftarrow$
m_{l+1}	$(m_l, m_{l-1}, \ldots, m_{l-M+1})$

- **At each time unit, the output block depends on the input and the state,**

$$\underline{V}_l = f(m_l, S_l) \qquad \textbf{(10)}$$

State Diagram (Pictorial Representation)

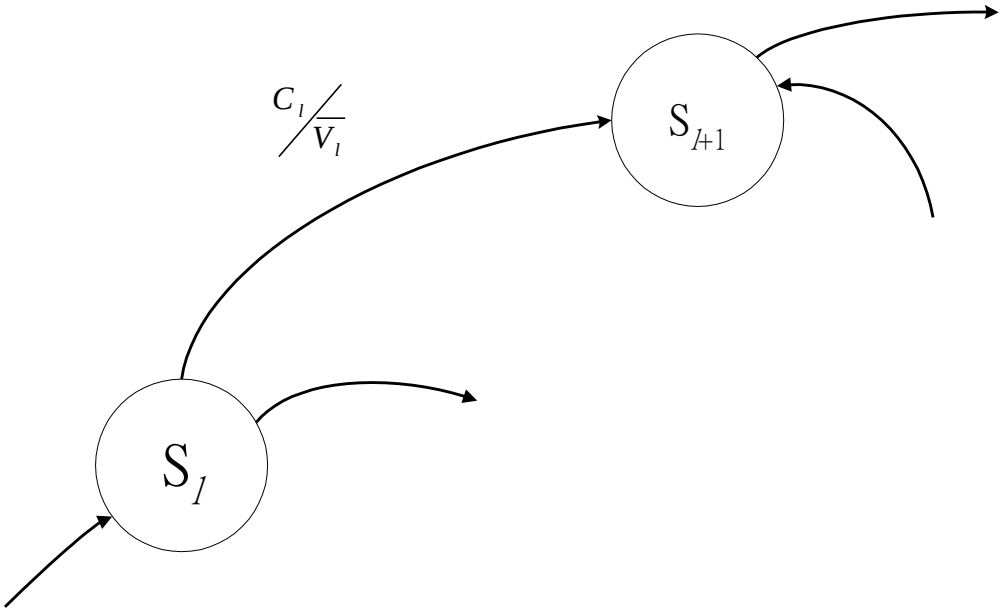
- **Each state is represented by a vertex (or point) on a plane.**
- **The transition from one state to another state is represented by a directed line (arc).**
- **Each directed line is labeled with I/O (input/output) pair.**

- **Suppose m_l is the current input. The current state of the encoder is $S_l = (m_{l-1}, m_{l-1}, \ldots, m_{l-M})$ (11)**

  When c_l is shifted into the encoder, the encoder moves into the state $S_{l+1} = (m_l, m_{l-1}, \ldots, m_{l-M+1})$ (12)

  which is called the next state.

- **The encoder is completely characterized by a state diagram, as shown in Fig.8.3**



- **Example 7.2:**

  Let n=2, k=1 and M=2. Consider a rate *1/2* (2,1,2) convolutional code which is specified by the following two generator sequences:

  $g^{(1)}(D) = 1+ D^2$ $\quad \overline{g}^{(1)} = (1\ 0\ 1),$

  $g^{(2)}(D) = 1+ D+ D^2$ $\quad \overline{g}^{(2)} = (1\ 1\ 1)$

  Let $m^{(i)} = (m_1^{(i)}, m_2^{(i)}, \ldots, m_j^{(i)}, \ldots)$ be the input message sequence. Then the two output sequence are:

$$\overline{C}^{(1)} = \overline{m} * \overline{g}^{(1)},$$
$$\overline{C}^{(2)} = \overline{m} * \overline{g}^{(2)}.$$

  – **The *l*-th output code block, , is given by**

$$c_l^{(1)} = m_l + m_{l-2}$$

$$c_l^{(2)} = m_l + m_{l-1} + m_{l-2}$$

The encoder shown in Fig.8.4 consists of 2 memory units.

- $V(D) = U(D)\, G(D)$

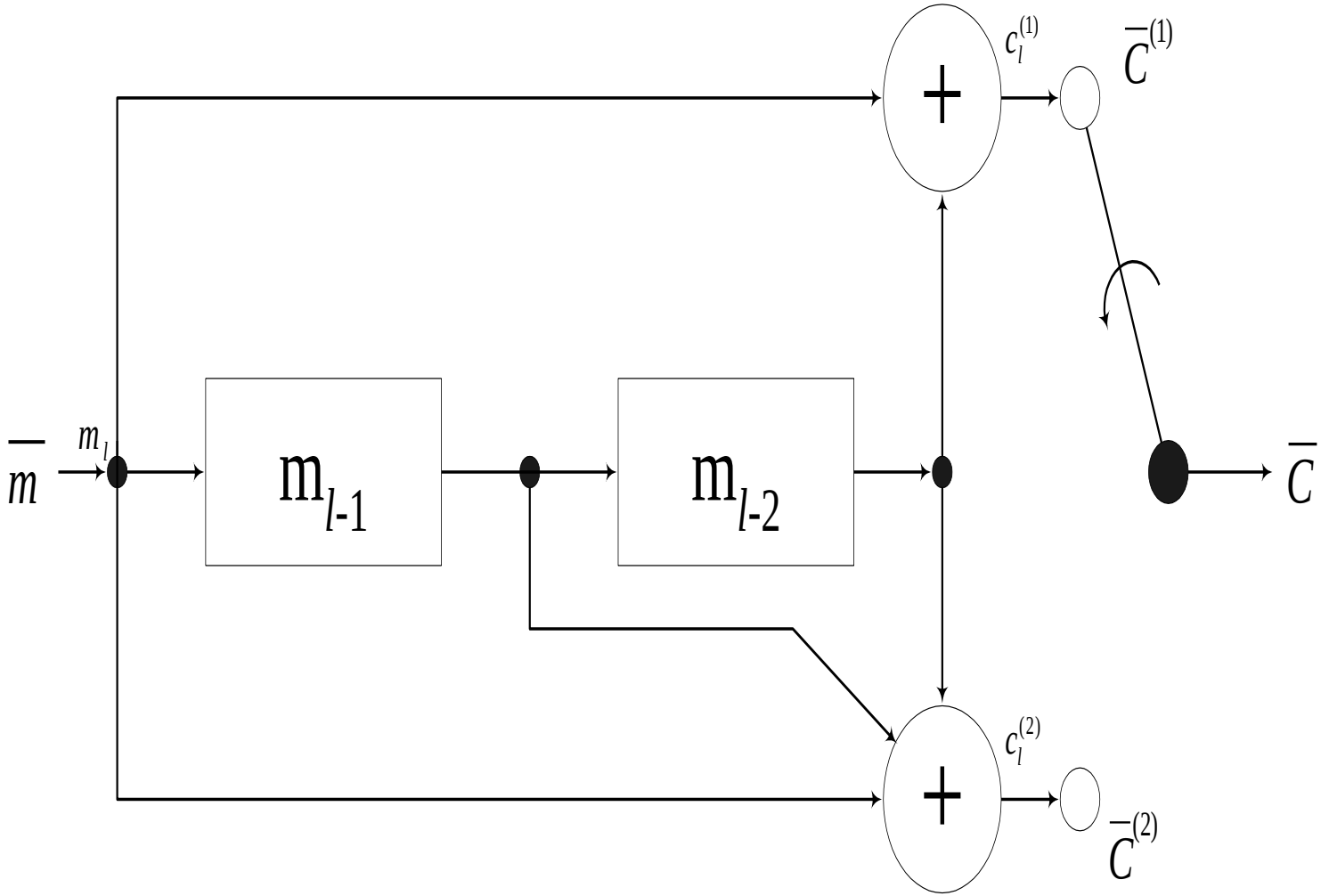
$$G(D) = \frac{(1+D^2)}{(1+D+D^2)}$$

$$U = m_l$$

$$V(D) = m_l + m_{l-2}$$

$$m_l + m_{l-1} + m_{l-2}$$

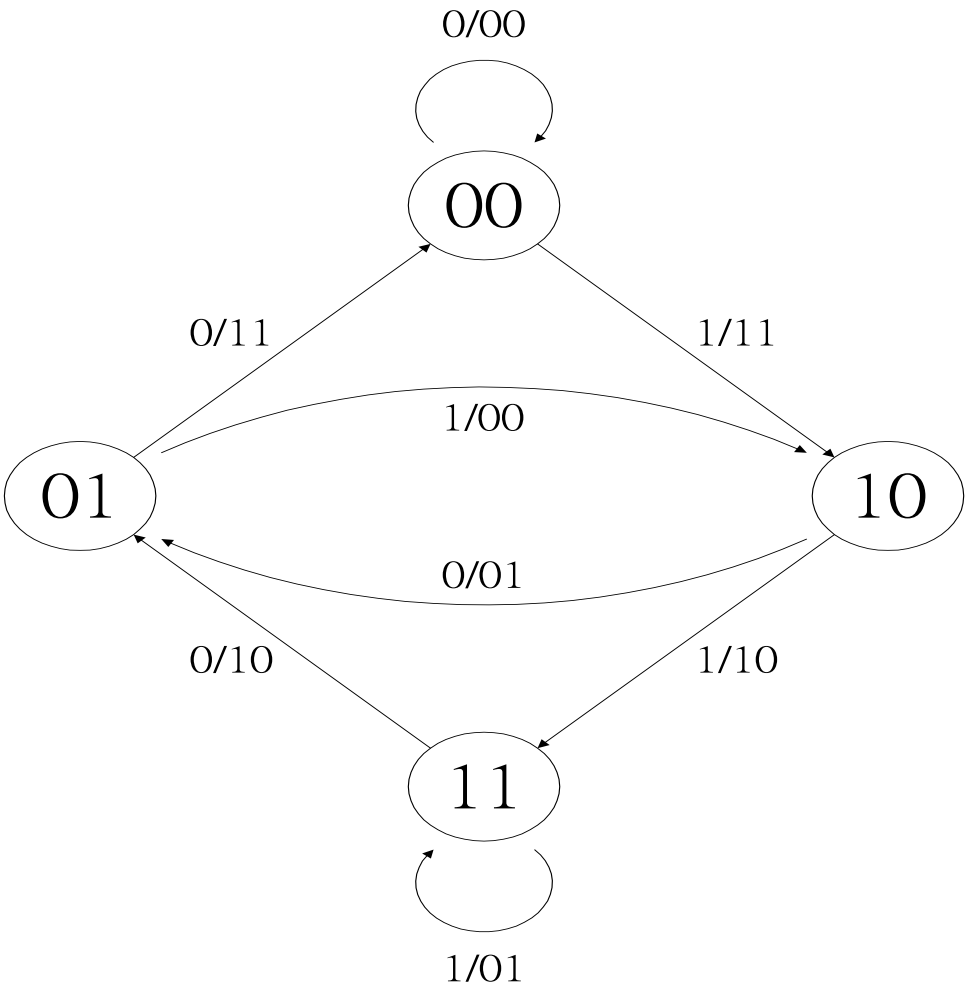
Fig.7.4


$\overline{m}$
m_l
m_{l-1}
m_{l-2}
+
+
$c_l^{(1)}$
$c_l^{(2)}$
$\overline{C}^{(1)}$
$\overline{C}^{(2)}$
$\overline{C}$

- **Each state is one of the forms:**

  (0.0), (0,1), (1,1), and (1,0).

  The state diagram is shown in Figure 7.5.



7.2.5 Trellis Diagram

- **The state diagram can be expanded in time to display the state transition of a convolutional encoder in time. This expansion in time results in a trellis diagram.**
- **Normally the encoder starts from the all-zero state,**

  (0, 0, …, 0).
- **When the first message bit m_1 is shifted into the encoder register, the encoder is in one of the two following states:**

  (m_1 = 0, 0, 0, …, 0) ; (m_1 = 1, 0, 0, …, 0);
- **When the second message bit m_2 is shifted into the encoder register, the encoder is in one of the following states:**

  (m_2 = 0, m_1= 0, 0, 0, …, 0); (m_2 = 1, m_1 = 0, 0, 0, …, 0);

  (m_2 = 0, m_1 = 1, 0, 0, …, 0); (m_2 = 1, m_1 = 1, 0, 0, …, 0);

- **Every time, when a message bit is shifted into the encoder register, the number of state is doubled until the number of states reaches 2^M.**

- **At the time M, the encoder reaches the steady state.**
- **At the time $l > M$, the encoder is in the state,**

$$(m_{l-1}, m_{l-2}, \ldots, m_{l-M}).$$

- **At the time $l+1$, the encoder can move into one of the following states:**

$$(m_l = 0, m_{l-1}, m_{l-2}, \ldots, m_{l-M+1}).$$
$$(m_l = 1, m_{l-1}, m_{l-2}, \ldots, m_{l-M+1}).$$

- **Therefore, in trellis diagram, there are two branches (or transitions) leaving a state.**
- **Now, suppose the state of the encoder is**

$$(m_l, m_{l-1}, m_{l-2}, \ldots, m_{l-M+1}). \quad \text{for } l > M.$$

- **This state can be reached from two states,**

$$(m_{l-1}, m_{l-2}, \ldots, m_{l-M+1} = 0)$$
$$(m_l, m_{l-1}, m_{l-2}, \ldots, m_{l-M+1}, m_{l-M} = 1)$$

- **Thus, for $l > M$, there are two branches merging into a state in the trellis diagram.**

Example 6.3:

- **Consider the (2,1,2) convolutional code given Example 8.2. Its trellis diagram is shown in Figure 8.6**
- **We see that there are two branches leaving each state, depending on the input symbol, $m_l = 0$ or $m_l = 1$.**
- **The upper branch corresponds to an input symbol $m_l = 1$, while the lower branch corresponds to an input symbol**

  $m_l = 0$.
- **For $l > M = 2$, we see that there are two branches merging into a state.**
- **The encoding of a message sequence is equivalent to tracing a path through the trellis.**

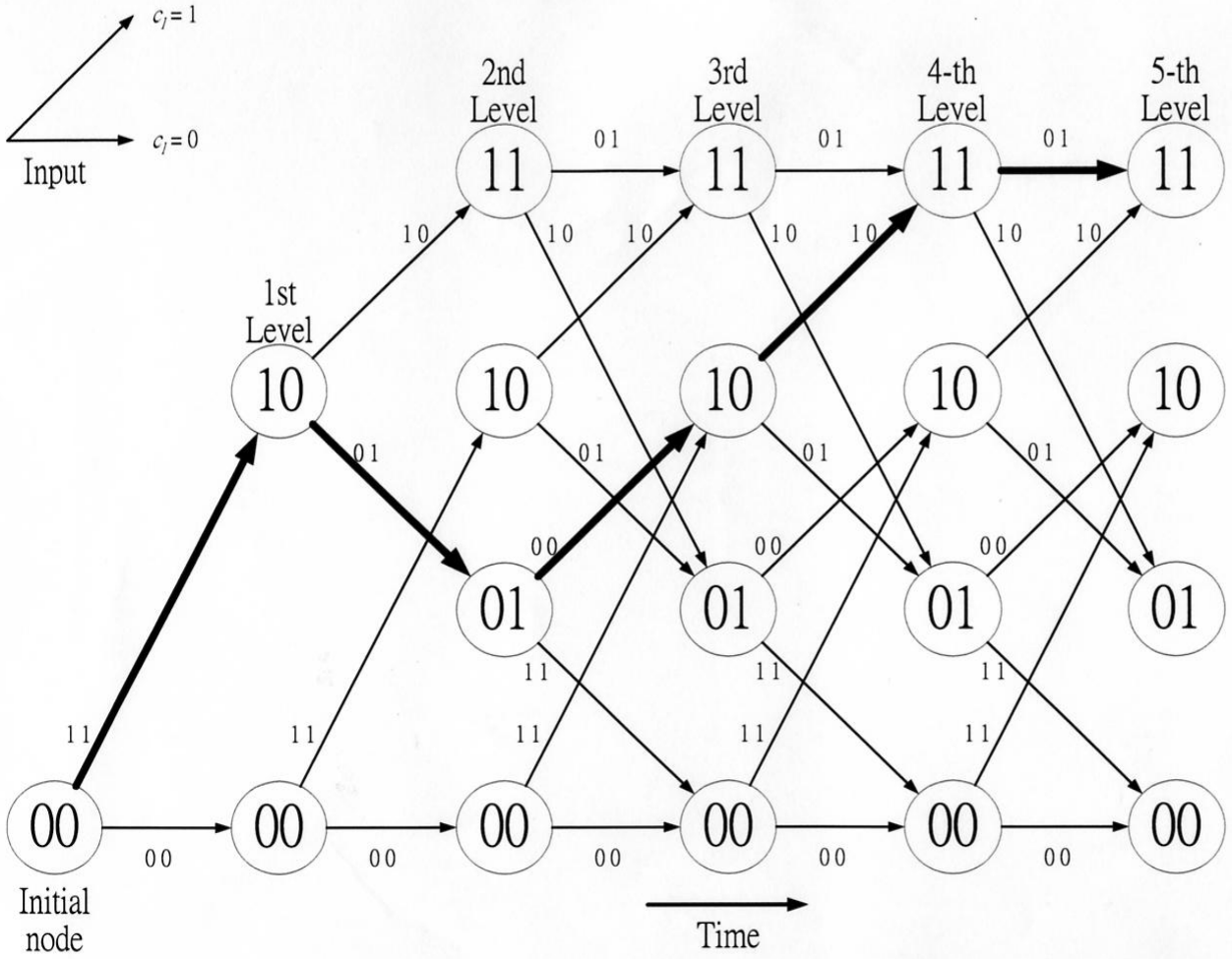


Figure 6 The trellis diagram for a (2,1,2) code

Termination of a Trellis

- Suppose the message sequence is of *L* bits long,

  $m = (m_1, m_2, \ldots, m_L)$

- When the entire sequence has been encoded, the encoder must return to the starting state. This can be done by appending *M* zeros to the message sequence *m* .

- When the first appended "0" is shifted into the encoder register, the encoder is in the state,

  $(0, m_{L-1}, m_{L-2}, \ldots, m_{L-M+1})$

- There are 2^{M-1} such states.

- **When the second "0" is shifted into the encoder register, the encoder is in the state,**

  $(0, 0, m_{L-1}, m_{L-2}, \ldots, m_{L-M+2})$

  There are 2^{M-2} such states.

- **When the *M*-th "0" is shifted into the register, the encoder is back to the all-zero state, (0, 0, …, 0).**

- **At this instant, the trellis converges into a single vertex.**

- **During the termination process, the number of states is reduced by half as each "0" is shifted into the encoder register.**

Example 7.4

Again, we consider the (2,1,2) convolutional code given in Example 7.2. The trellis diagram corresponds to a message sequence of 5 bits long.

The trellis has a depth of 7 as shown in Figure 7.7

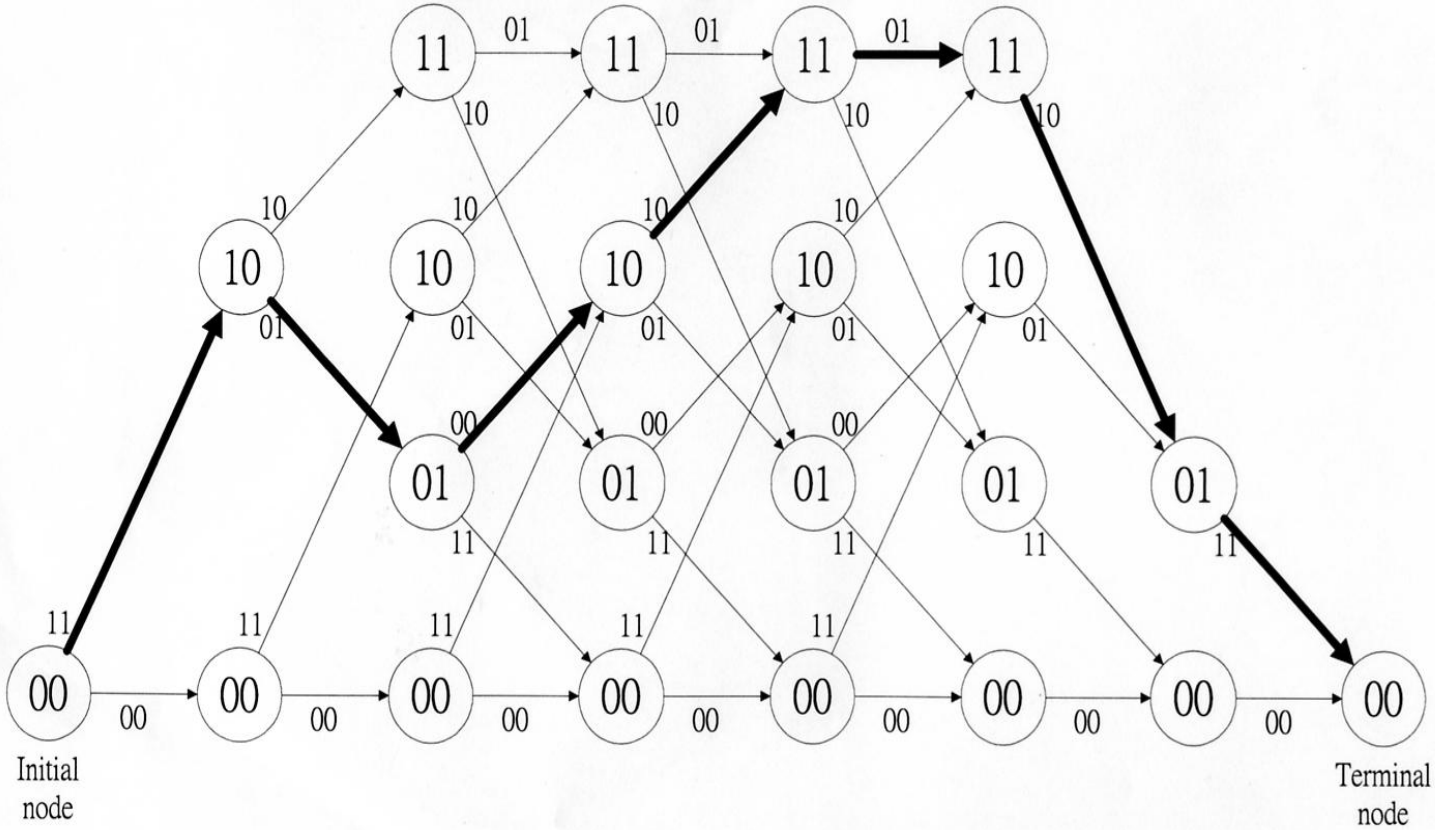


Figure 7 A terminated trellis diagram with $L = 7$

7.2.6 Minimum Free Distance

- **The most important distance measure for convolutional codes is the minimum free distance, denoted d_{free}.**
- **The minimum free distance of a convolutional code is simply the minimum Hamming distance between any two code sequences in the code.**
- **It is also the minimum weight of all the code sequences, which are produced by the nonzero message sequences.**
- **The minimum free distance of the (2,1,2) convolutional code given in Example 8. 2 is 5 , i.e., $d_{free}=5$.**

Summary :

- **A (*n.k,M*) convolutional code can be represented by :**
  1. **Encoder block diagram using digital circuits (shift registers , adders , etc.)**
  2. **Generator polynomials, $g^{(i)}(D)$.**
  3. **State diagram**
  4. **Trellis diagram**

  Note : constraint length $K = M+1$

  number of states = 2^M

The Most Widely Used Convolutional Codes

- **The most widely used convolutional code is (2,1,6) Odenwalter code generate by the following generator sequence,**

  $g^{(1)}(D) = (1\ 111\ 001)$

  $g^{(2)}(D) = (1\ 011\ 011)$

  This code has d_{free}=10.

- **With hard-decision decoding, it provides a 3.98dB coding gain over the uncoded BPSK modulation system.**

  With soft-decision decoding, the coding gain is 6.98dB.

7.3 Maximum Likelihood Decoding of Convolutional Codes

7.3.1 Maximum Likelihood Decoding

- For a convolutional code, each code sequence is a path in the trellis diagram of the code.

- Suppose each message sequence consists of L message blocks of k bits each, $\mathbf{m} = (m_1, m_2, \ldots, m_L)$

  .

- Then each code sequence $\mathbf{c}$ is a path of $L+M$ branches long in the trellis diagram, $\mathbf{c} = (c_1, c_2, \ldots, c_L)$

  where the l-th branch (or code block)

  $c_l = (v_1^{(1)}, v_2^{(2)}, \ldots, v_L^{(n)})$

- Suppose a code sequence is transmitted.

  Let $\mathbf{c} = (c_1, c_2, \ldots, c_L)$

  be the received sequence where the l-th received block. .

- **MLD: Find the path through the trellis diagram such that the conditional probability, $P(\mathbf{r} \mid \mathbf{c})$ is the largest.**

- **For a binary input, *Q*-ary output discrete memoryless channel (DMC), $\bar{c}$ is a binary sequence and $\bar{r}$ is a *Q*-ary sequence.**
- **The conditional probability $P(\bar{r} \mid \bar{c})$ can be computed as follows:**

$$P(\bar{r} \mid \bar{c}) = \prod_{l=0}^{L+m-1} P(\bar{r}_l \mid \bar{c}_l) \qquad (13)$$

where $P(\bar{r}_l \mid \bar{c}_l)$ is the branch conditional probability.

- **The branch conditional probability is given by**

$$P(\bar{r}_l \mid \bar{c}_l) = \prod_{i=0}^{n} P(r_l^{(i)} \mid c_l^{(i)}) \qquad (14)$$

where $P(r_l^{(i)} \mid c_l^{(i)})$ is the channel transition probability.

- **Define the log-likelihood function of a path c as follows:**

$$M(\bar{r} \mid \bar{c}) \equiv \log P(\bar{r} \mid \bar{c}) \qquad (15)$$

which is called the metric of path .

- From (13) and (15), we have

$$M(\bar{r} \mid \bar{c}) \equiv \sum_{l=0}^{L+m-1} \log P(\bar{r}_l \mid \bar{c}_l) = \sum_{l=0}^{L+m-1} M(\bar{r}_l \mid \bar{c}_l) \qquad (16)$$

where $M(\bar{r}_l \mid \bar{c}_l) \equiv \log P(\bar{r}_l \mid \bar{c}_l)$ (17)

is called the **branch metric**.

- From (14) and (17), we have the branch metric

$$M(\bar{r}_l \mid \bar{c}_l) \equiv \sum_{i=1}^{n} \log P(r_l^{(i)} \mid c_l^{(i)}) \qquad (18)$$

where

$$M(r_l^{(i)} \mid c_l^{(i)}) = \log P(r_l^{(i)} \mid c_l^{(i)}) \qquad (19)$$

is called the **bit metric**.

- MLD: Find the path $\bar{c}$ in the trellis diagram such that $M(\bar{r} \mid \bar{c})$ is maximized. Then $\bar{c}$ is the estimate of the transmitted code sequence.
- For the first j branches of a path through the trellis, the partial path metric is

$$M([\bar{r} \mid \bar{c}]_j) = \sum_{l=0}^{j-1} M(\bar{r}_l \mid \bar{c}_l) \qquad (20)$$

7.3.2 Maximum Likelihood Decoding for a BSC

- **For a BSC (Q=2) with transition probability $p < \frac{1}{2}$, the log-likelihood function becomes**

$$\log P(\bar{r} \mid \bar{c}) = d(\bar{r}, \bar{c}) \log \frac{p}{1-p} + (L+m)n \log(1-p) \qquad (21)$$

where $d(\bar{r}, \bar{c})$ is the Hamming distance between $\bar{r}$ and $\bar{c}$

Since $\log[p/(1-p)] < 0$ and $(L+m)n \log(1-p)$ is a constant for all code sequences , $\log P(\bar{r} \mid \bar{c})$ is maximized if and only if $d(\bar{r}, \bar{c})$ is minimized.

- **MLD: The received sequence $\bar{r}$ is decoded into the code sequence $\bar{c}$ if $d(\bar{r}, \bar{c})$ is minimized.**

7.3.3 The Viterbi Decoding Algorithm

- **The Viterbi algorithm performs maximum likelihood decoding but reduces the computational complexity by taking advantage of the special structure of the code trellis.**
- **It was first introduced by A. Viterbi in 1967 and was first recognized by D. Forney in 1973 that it is a MLD algorithm for convolutional code.**

The Viterbi Algorithm

- **Step 1. Starting at the level $l = m$ in the trellis, compute the partial metric for the single path entering each m-th order node. Store the path (the survivor) and its metric for each node.**
- **Step 2.**

  Increasing l by 1.

  Compute the partial metric for all the paths entering a (l+1)-th order node by adding the branch metric entering that node to the metric of the connecting survivor at a previous l-th order node.

  For each (l+1)-th node, store the path with the largest metric (the survivor) , together with its metric, and eliminate all the other paths.
- **Step 3. If $l < L+M$, repeat Step2. Otherwise, stop.**

Example 7.5

- **Consider the (2,1,2) convolutional code given in Example 7.2 whose trellis diagram is shown in Fig. 7.1. Suppose the code is used for a BSC. In this case, we may use the Hamming distance as the path metric. The survivor at each node is the path with the smallest Hamming distance from the received sequence.**
- **The message length $L = 5$.**
- **There are 7 levels in the trellis.**
- **The received words are 01 11 10 10 00 11 10**
- **The decoding process is shown in Figures 8.9 to 8.15.**

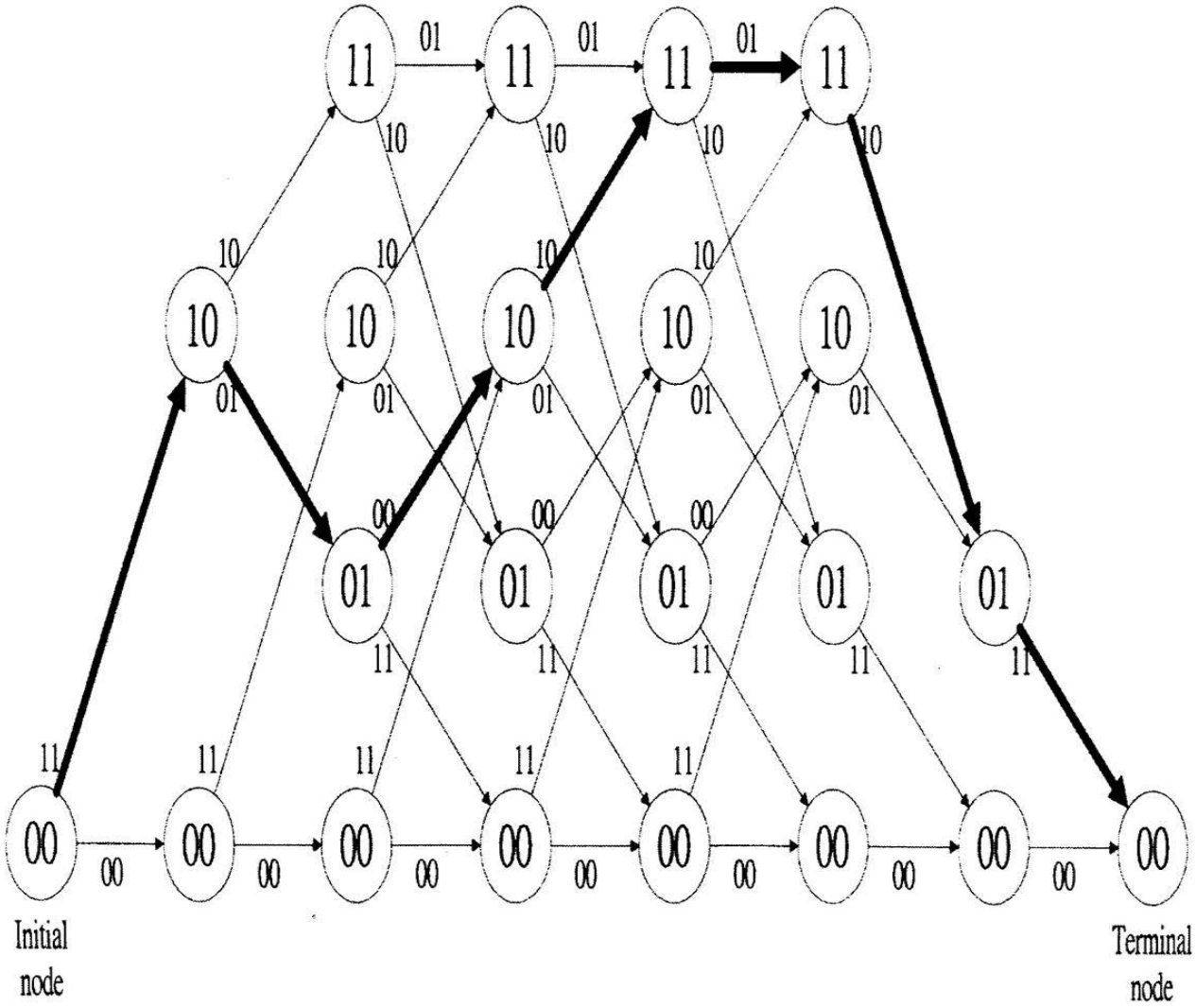


Fig. 4.10 The trellis diagram of a (2,1,2) code with $L = 5$.

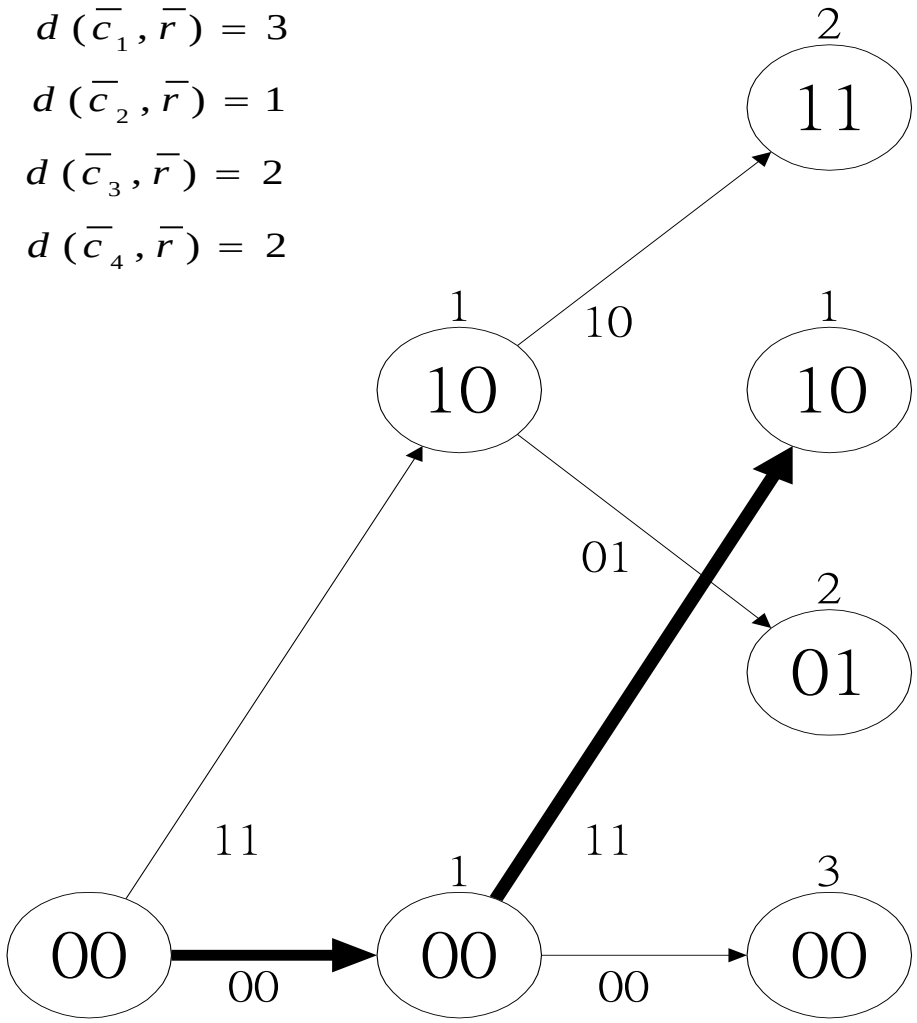
Figure 7.9 Decoding process at level 2,

$\bar{c}_1 = (00, 00), \quad d(\bar{c}_1, \bar{r}) = 3$

$\bar{c}_2 = (00, 11), \quad d(\bar{c}_2, \bar{r}) = 1$

$\bar{c}_3 = (11, 01), \quad d(\bar{c}_3, \bar{r}) = 2$

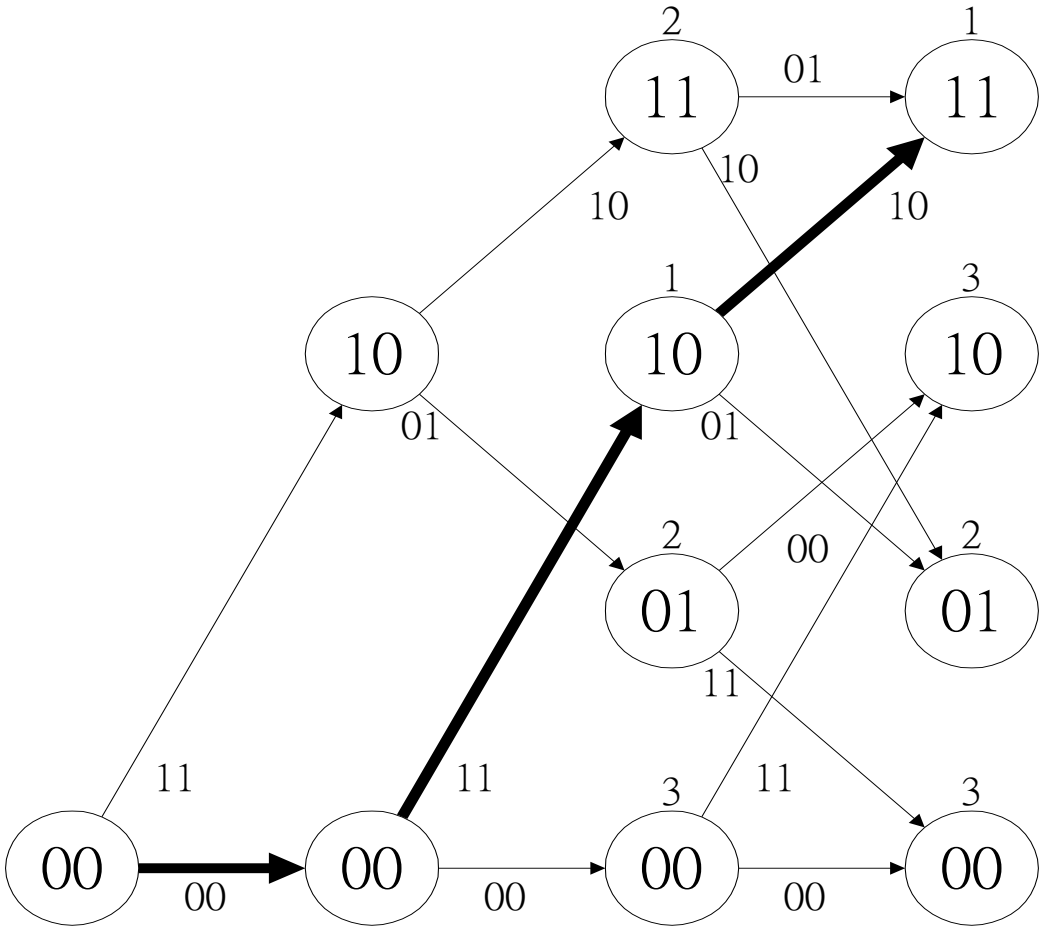
$\bar{c}_4 = (11, 10), \quad d(\bar{c}_4, \bar{r}) = 2$



$\bar{r} = (01, 11)$

Figure 7.10

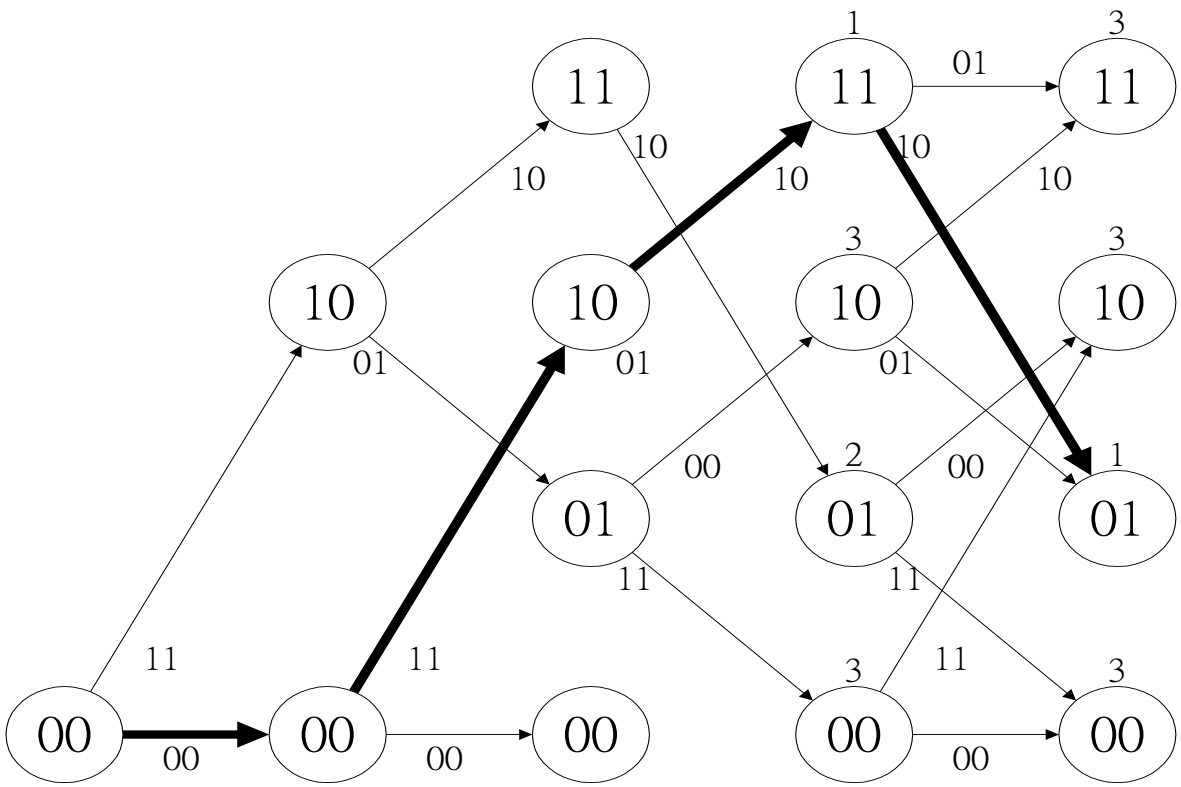
Decoding process at level 3 (comparison and elimination)



$$\bar{r} = (01, 11, 10)$$

Figure 7.11

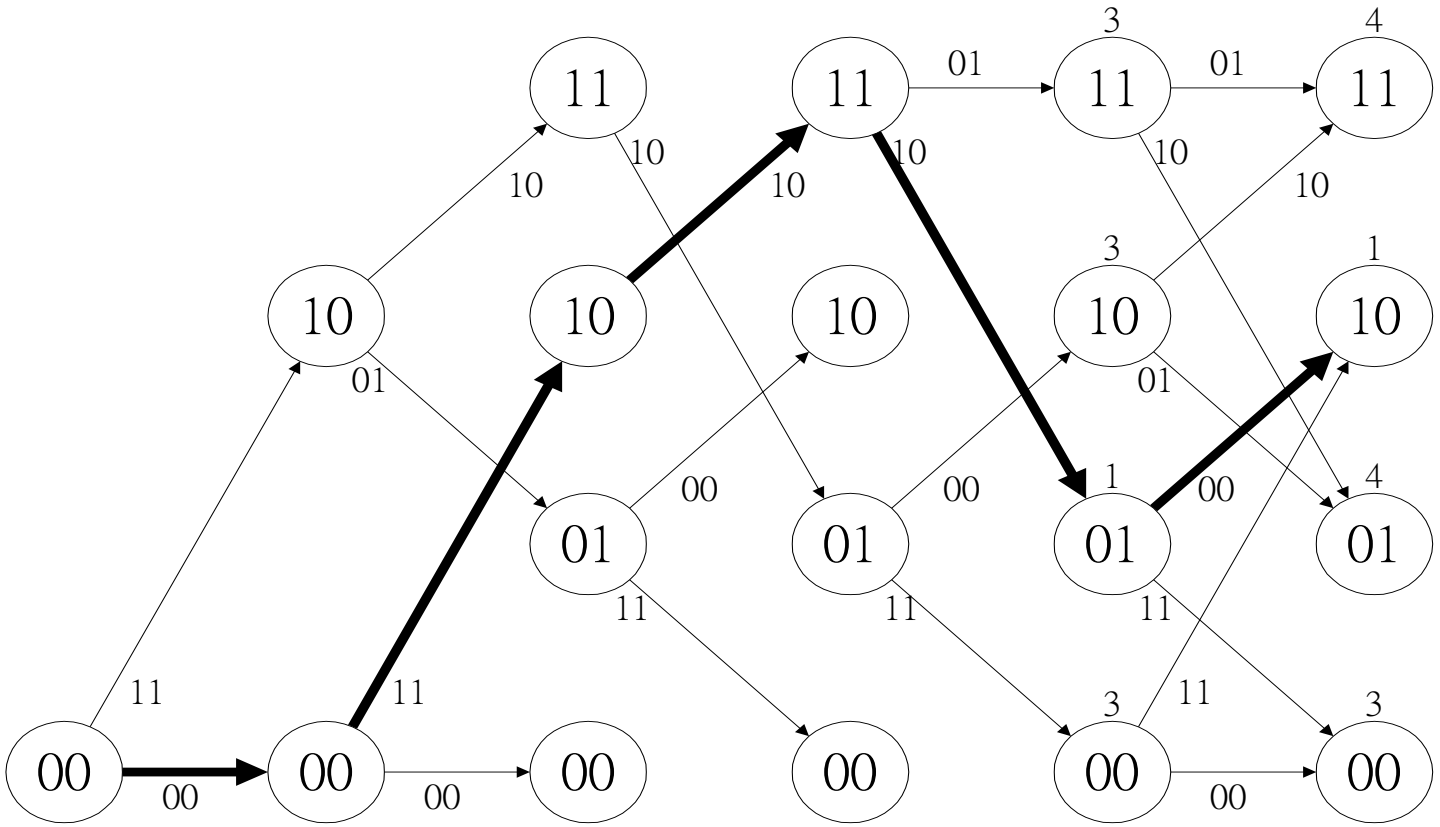
Decoding process at level 4 (comparison and elimination)



$$\overline{r} = (01, 11, 10, 10)$$

Figure 7.12

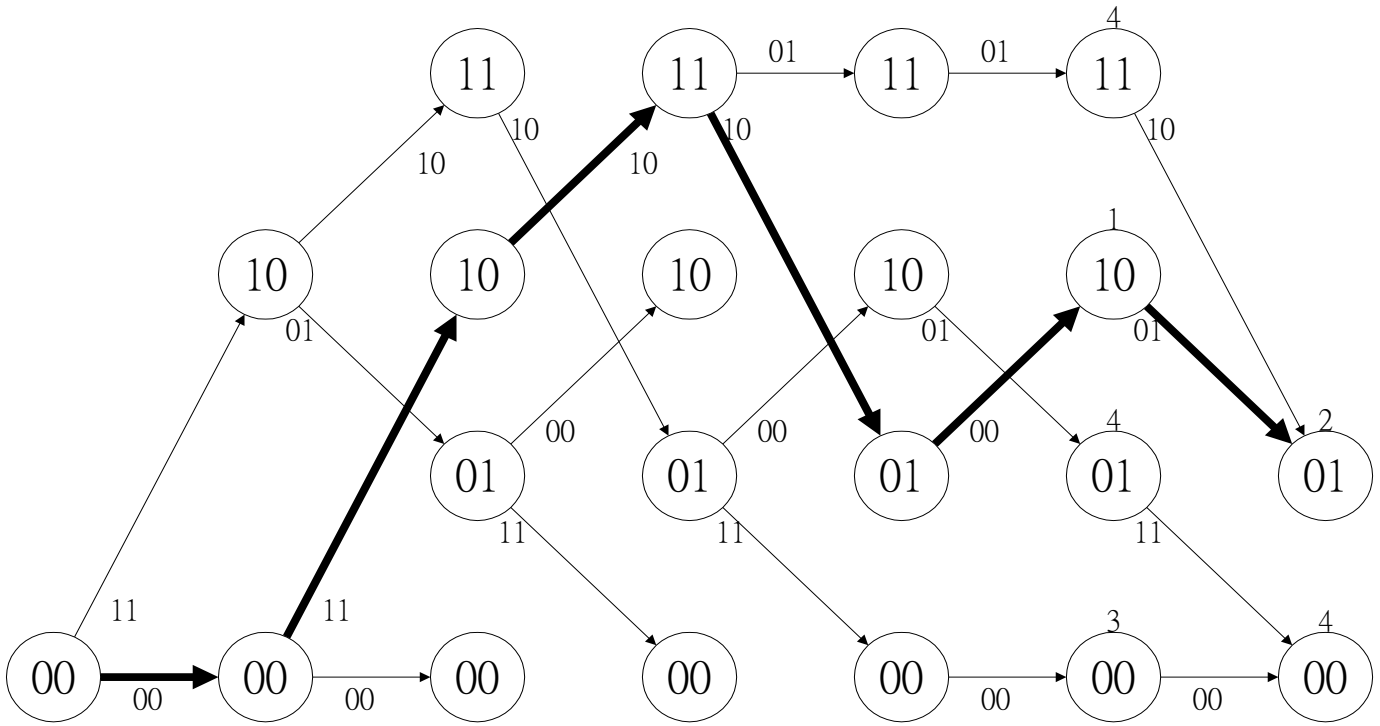
Decoding process at level 5 (comparison and elimination)



$$\overline{r} = (01, 11, 10, 10, 00)$$

Figure 7.13

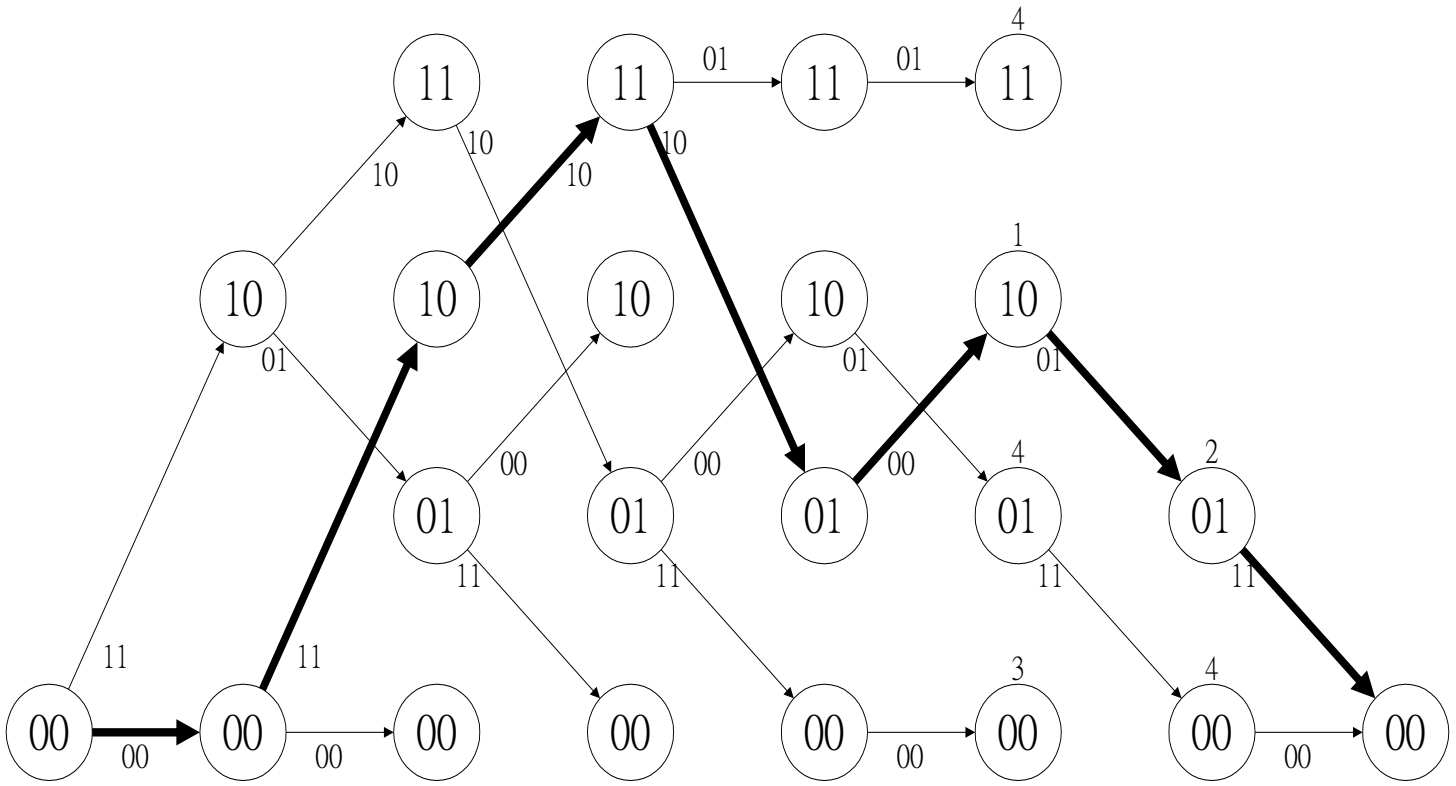
Decoding process at level 6 (comparison and elimination)



$$\overline{r} = (01, 11, 10, 10, 00, 11)$$

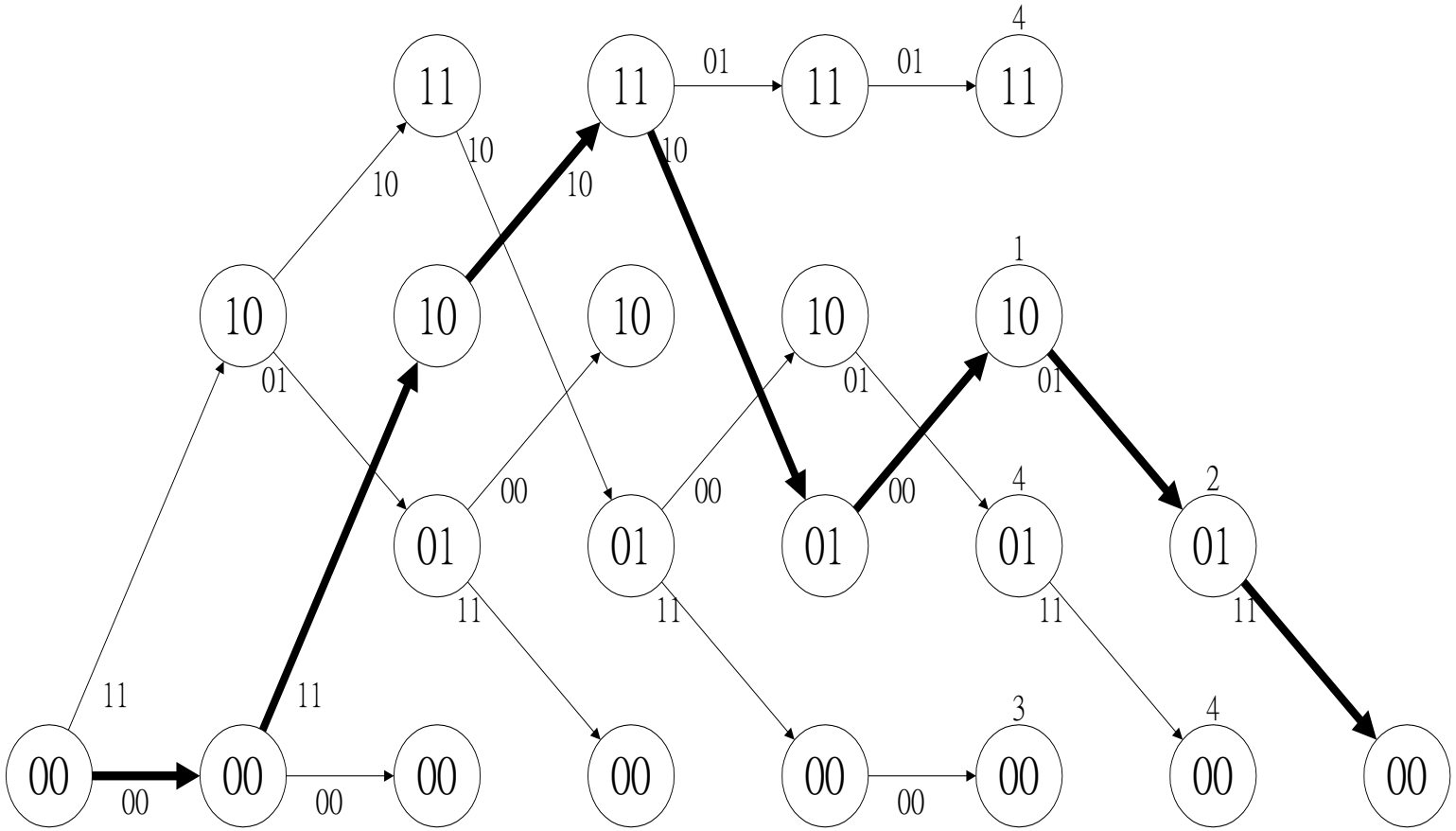
Figure 7.14

Decoding process at level 7 (comparison and elimination)



$$\overline{r} = (01, 11, 10, 10, 00, 11, 10)$$

Figure 7.15 Decoding termination



7.3.4 Coding Gain

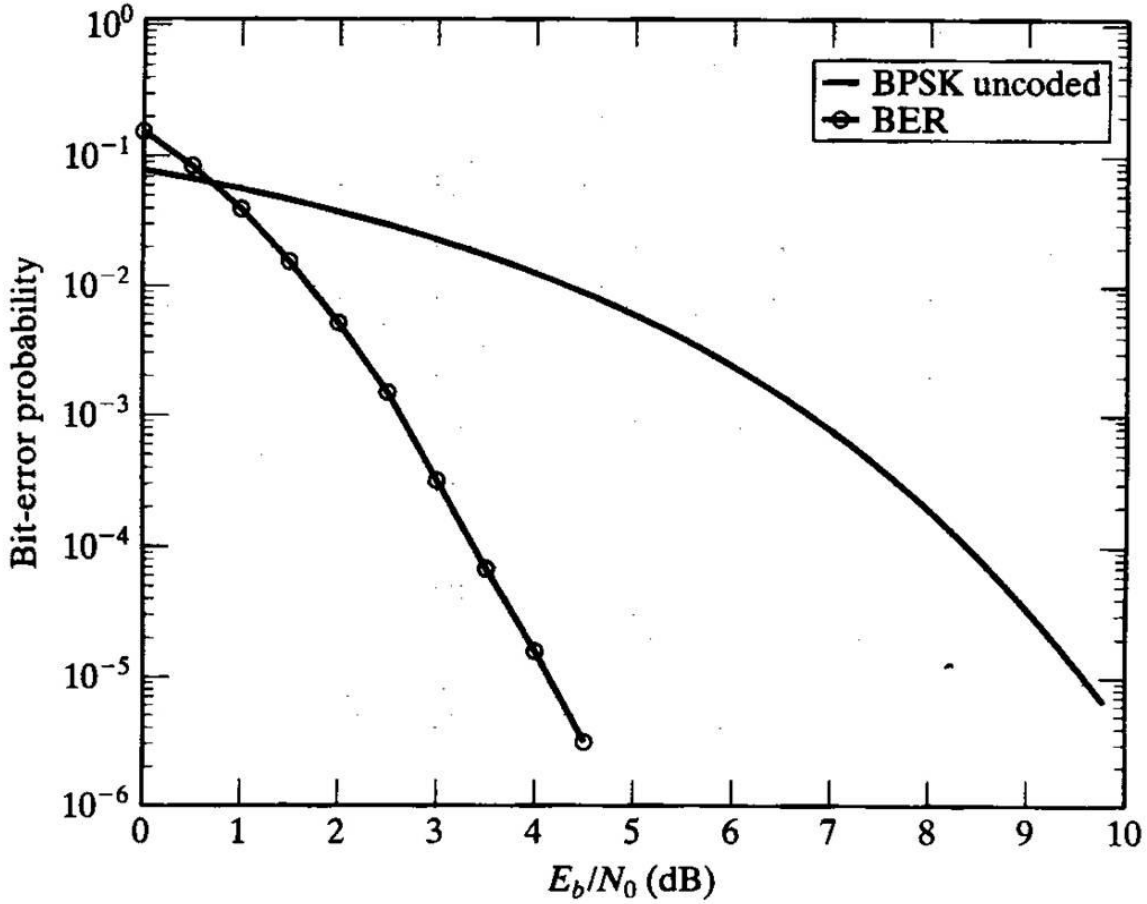
- **Coding gain is defined as the reduction in the require E_b/N_0 (usually expressed in decibels) to achieve a specified error probability of a coded system over an uncodeed system with the same modulation and channel characteristic , as illustrated in Fig.8.16.**
- **For an uncoded coherent BPSK system with an AWGN channel, the bit-error rate simply the transition probability,**

$$P_b(E) = Q\left(\sqrt{\frac{2E}{N_0}}\right)$$

- **For large E_b/N_0, this error rate (without coding) is approximated by**

$$P_b(E) \approx 0.282\ e^{-E_b/N_0} \qquad \textbf{(22)}$$

Fig.7.16 Illustration of coding gain



Modifications of Viterbi Algorithm : Truncation

- **For very large *L*, this is practically impossible, and some trade-offs must be made.**
- **One approach to this problem is to truncate the path memory of the decoder by storing only the most recent *r* blocks of message bits for each survivor, where $r << L$.**
- **After the first *r* blocks of the received sequence have been processed by the decoder, the decoder memory is full.**
- **As soon as the next received block is processed, a decoding decision must be made on the first block of *k* message bits, since they can no longer be stored in the decoder memory.**

- **The optimum strategy to make this decision is to select the survivor with the best metric, and the first block of k message bits of this survivor is chosen as the decoded message block and released to the user.**

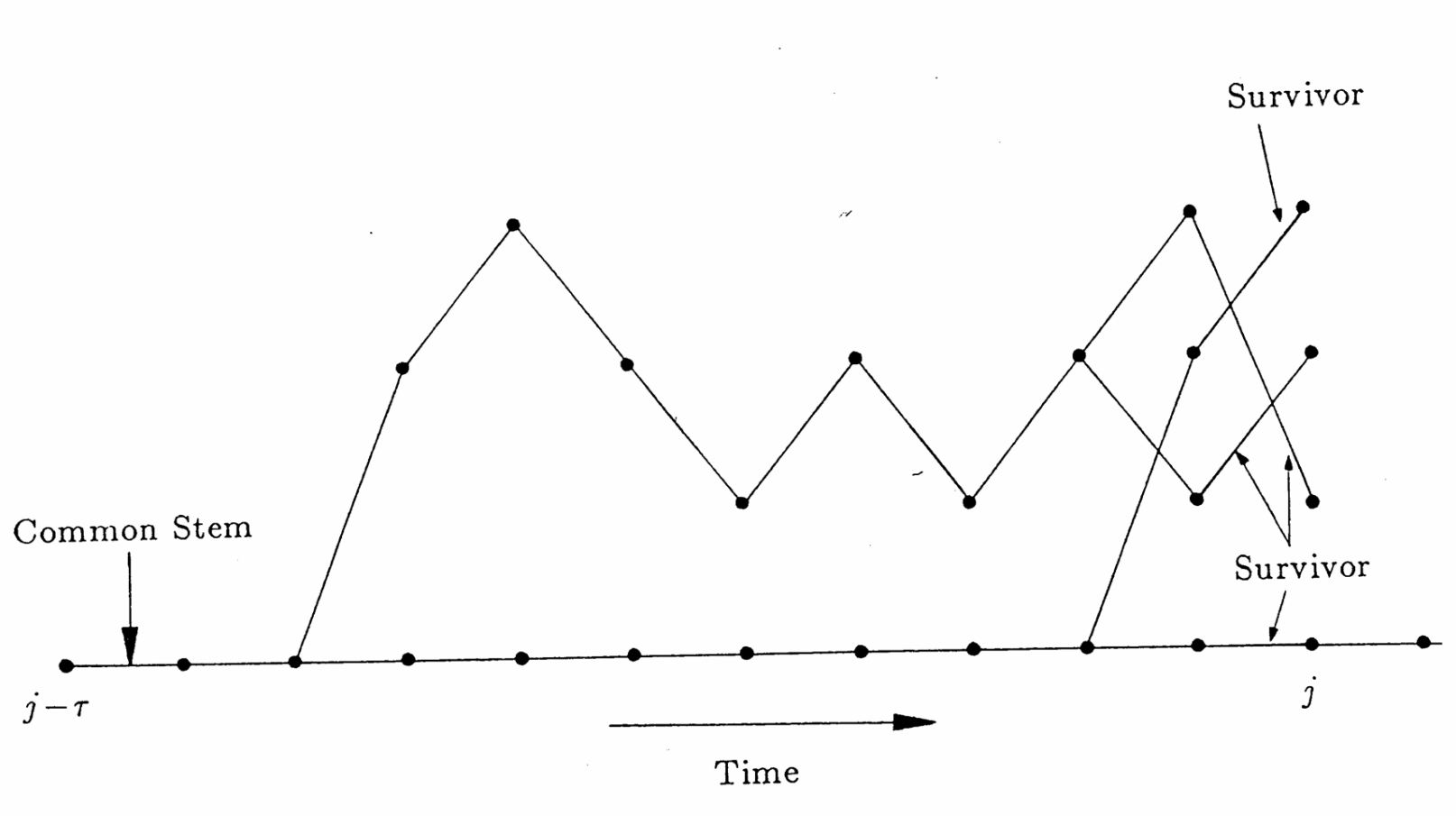
- **After the first decoding decision is made, subsequent decoding decisions are made in the same manner for each new received block processed.**
- **Note that decoding decisions made in this way are no longer maximum likelihood, but can be almost as good if *r* is large enough.**

- **Experience and analysis have shown that if *r* is in the order of 5 times of the encoder memory *K* or more, with probability approaching "1", all the 2*K* survivors stem from the same branch *r* levels back as shown in Figure 8.17.**

- **Hence there is no ambiguity in making decoding decision. The parameter *r* is called the decoding span (or depth).**

Figure 7.17

Decoding decision with a finite path memory *r*



7.4 Punctured Convolutional Codes

- **In many bandwidth-limited application s , high-rate or low-redundancy convolutional codes are desirable . However, the Viterbi decoder forthese codes is often quite complicated . For the (n,k.M) binary convolutional code , the complexity of the Viterbi decoding is proportional to 2^{kM} .**
- **Puncturing is a technique used to make a *k/n* rate code from a "basic" rate 1/2 code. It is reached by deletion of some bits in the encoder output. Bits are deleted according to *puncturing matrix*.**
- **This has the same effect as encoding with an error-correction code with a higher rate, or less redundancy. However, with puncturing the same decoder can be used regardless of how many bits have been punctured, thus puncturing considerably increases the flexibility of the system without significantly increasing its complexity**

- **A pre-defined pattern of puncturing is used in an encoder. Then, the inverse operation, known as depuncturing, is implemented by the decoder. Puncturing is often used with the Viterbi Algorithm in coding systems.**

7.4.1 Rate R=*2/3* Punctured Convolutional Code

- **We are to make a code with rate 2/3 using the 4-state ,**

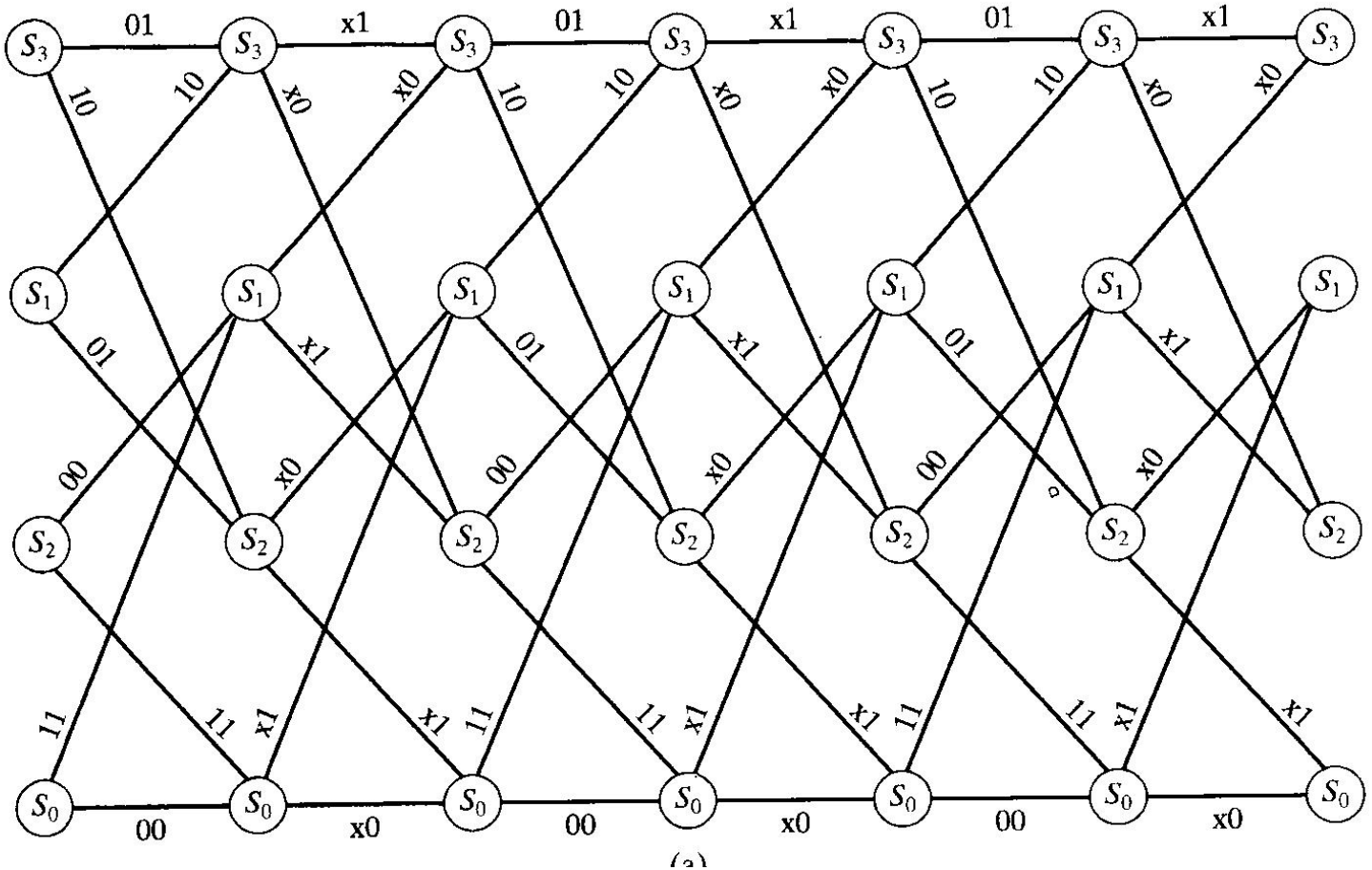
  rate $R = 1/2$ **mother code generated by the (2,1,2) non-systematic feedforward convolutional encoder with the generator matrix**

  $$G(D) = [\, 1+D^2 \,,\, 1+D+D^2 \,]$$

  This code has free distance $d_{free} = 5$ **.**

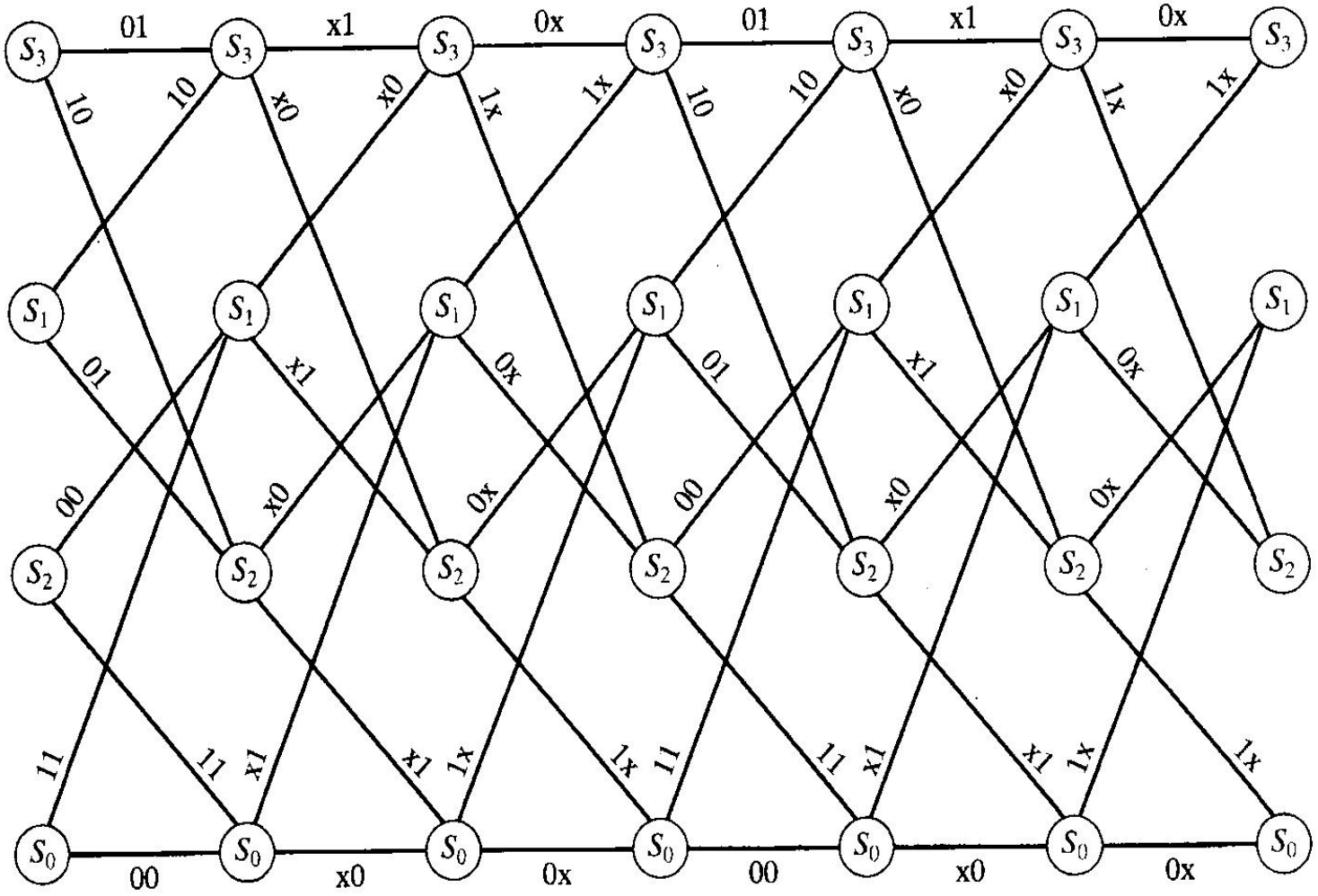
  we should take a basic encoder output and transmit every second bit from the first branch and every bit from the second one, that is, deleting the first bit on every other branch of the trellis, as shown in Fig, 8.18

Fig.7.18 A rate 2/3 punctured code



- A rate 3//4 punctured code

puncture vector parameter : 1 1 0 1 1 0



Puncturing Pattern of 2/3 and ¾ codes

- In each of the above case, the puncturing pattern is indicated using a 2 x *T* binary matrix P , where *T* is the puncturing period. The first row of P indicates the bits to be deleted from the first encode sequence , and the second row of P indicates the bits to be deleted from the second encoded sequence. In the matrix P , a 0 indicates a bit to be deleted , and a 1 indicated a bit to be transmitted.
- The puncturing patterns in Figure 12.23 are given by

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

Fig. (a)

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

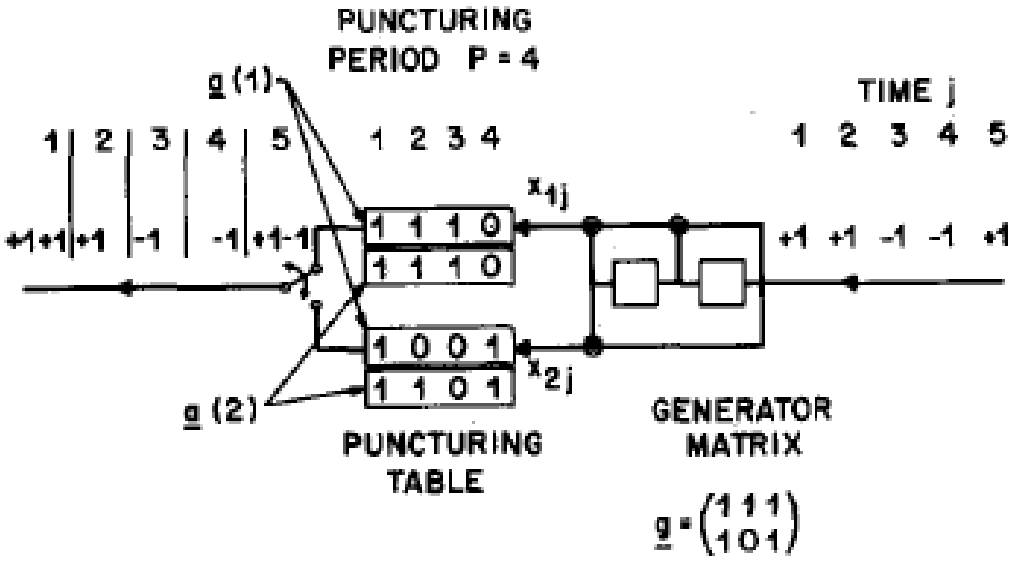
Fig. (b)

7.4.2 Rate-Compatible Punctured Convolutional (RCPC) Codes

- In applications where it is necessary to support two or more different code rates, it is sometimes convenient to make use of rate-compatible punctured convolutional (RCPC) codes.
- An RCPC code is a set of two or more convolutional codes punctured from the same mother code in such a way that the codewords of a higher-rate code can be obtained from the codewords of a lower-rate code simply by deleting additional bits.
- In other words, the set of puncturing patterns must be such that the P matrix of a higher-rate code is obtained from the P matrix of a lower-rate code by simply changing some of the 1's to 0's .
- An RCPC code then has the property that all the codes in the set have the same encoder and decoder.

Example of RCPC Codes

- R = ½, memory M = 2 ,convolutional code , punctured periodically with P=4



Example of a punctured convolutional code with two rate compatible puncturing tables.

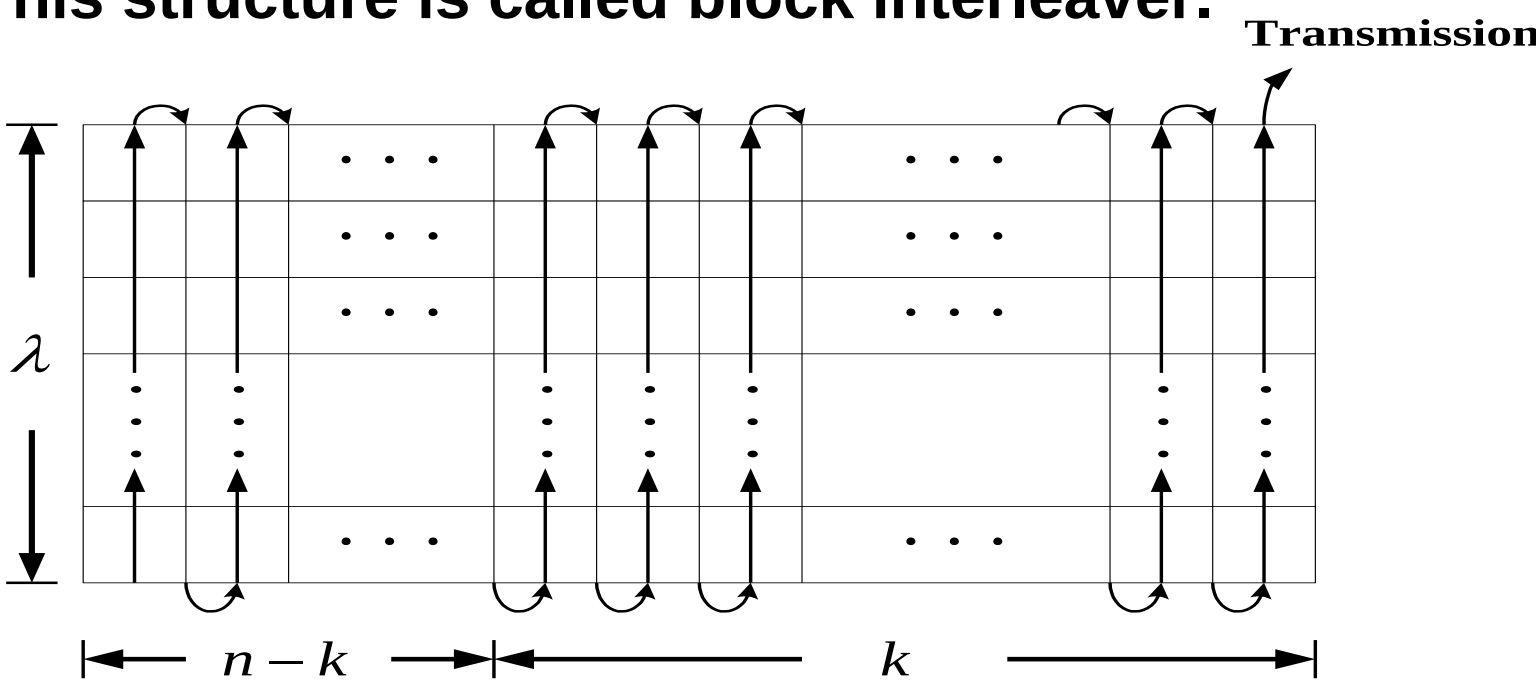
$$a(1)=\begin{pmatrix}1&1&1&0\\1&0&0&1\end{pmatrix}\quad a(2)=\begin{pmatrix}1&1&1&0\\1&1&0&1\end{pmatrix},\ a(3)=\begin{pmatrix}1&1&1&1\\1&1&0&1\end{pmatrix}\quad a(4)=\begin{pmatrix}1&1&1&1\\1&1&1&1\end{pmatrix}$$

7.5 Interleaving Technique and Concatination

7.5.1 Interleaving

Let C be an (n, k) linear code.

- **Suppose we take λ code words from C and arrange then intoλrows of anλ× n array as shown in the following figure. This structure is called block interleaver.**
-



- Then we transmit this code array column by column in serial manner. By doing this, we obtain a vector of λn digits.
- Note that two consecutive bits in the same codeword are now separated by λ-1 positions.
- Actually, the above process simply interleavesλcodewords in C. The parameter$\lambda$is called interleaving degree (or depth).
- There are $(2k)\lambda = 2k\lambda$ such interleaved sequences and they form a $(\lambda n, \lambda k)$ linear code, called an interleaved code, denoted $C(\lambda)$.
- If the base code C is a cyclic code with generator polynomial , then the interleaved code $C(\lambda)$ is also cyclic.

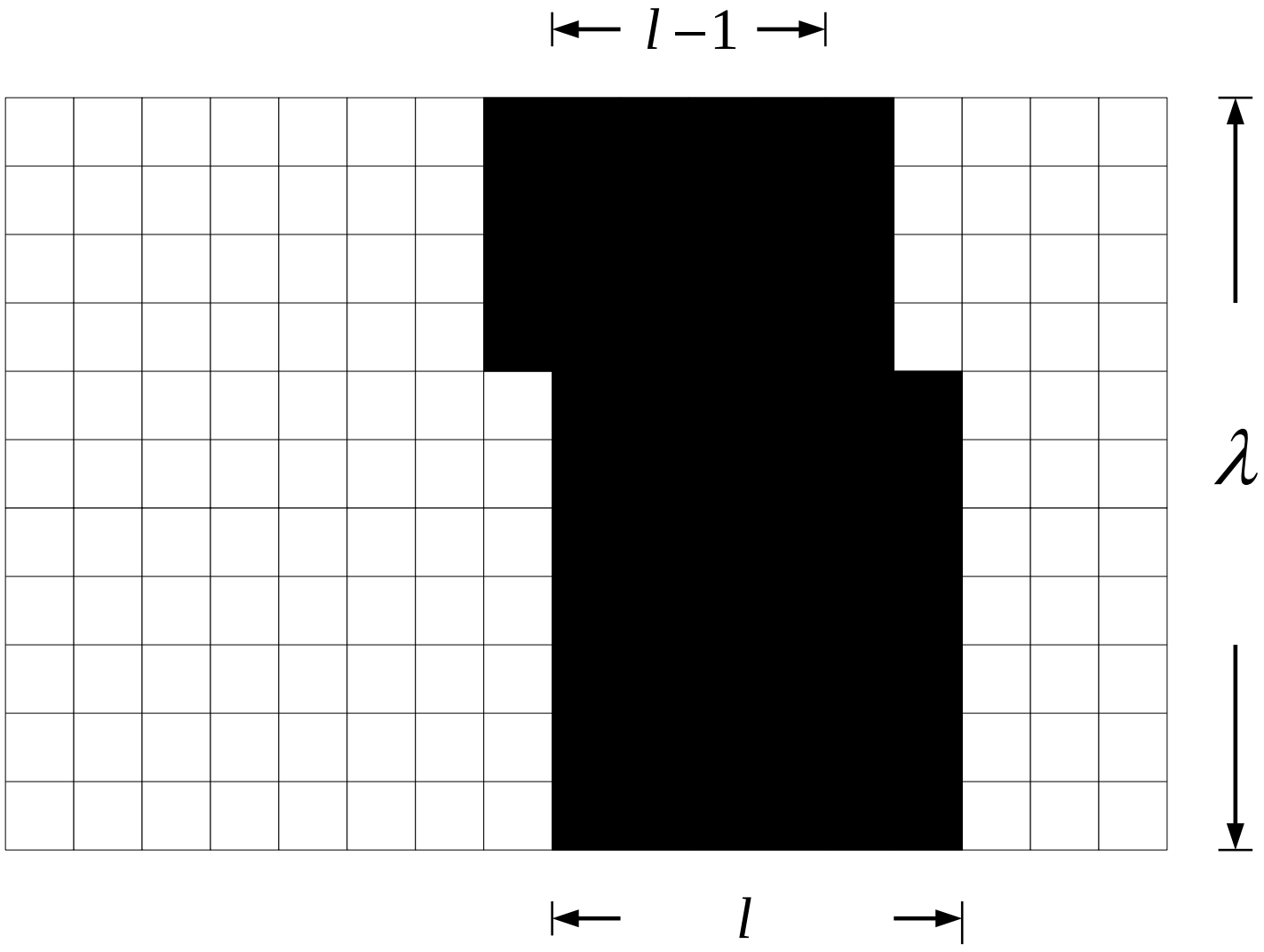
Error Correction Capability of an Interleaved Code

- A pattern of errors can be corrected for the whole array if and only if the pattern of errors in each row is a correctable pattern for the base code C.
- Suppose C is s single-error-correcting code.

  Then a burst of length λ or less, no matter where it starts, will affect no more than one digital in each roe. This single bit error in each row will be corrected by the base code C.
- Hence the interleaved code C(λ) is capable of correcting any error burst of lengthλ or less.

7.5.2 Decoding of Interleaved Code

- **At the receiving end, the received interleaved sequence is de-interleaved and rearranged back to a rectangular array of λ rows.**
- **Then each row is decoded based on the base code C.**
- **Suppose the base code C is capable of correcting any burst of length *l* or less.**
- **Consider any burst of length*λl* or less. No matter where this burst starts in the interleaved code sequence, t will result a burst of length *l* or less in each row of the corresponding code array as shown in the following Figure 4.7.2**


l −1
λ
l

- **As a result, the burst in each row will be corrected by the base code C.**

  Hence the interleaved code C(λ) is capable of correcting any single error burst of length λl or less.
- **Interleaving is a very effective technique for constructing long powerful burst-error correcting codes from good short codes.**
- **If the base code is an optimal burst-error-correcting code, the interleaved code is also optimal. Convolutional Interleaver：**
- **A convolutional interleaver can be used in place of a block interleaver in much the same way.**
- **Convolutional interleavers are better matched for use with the class of convolutional codes.**

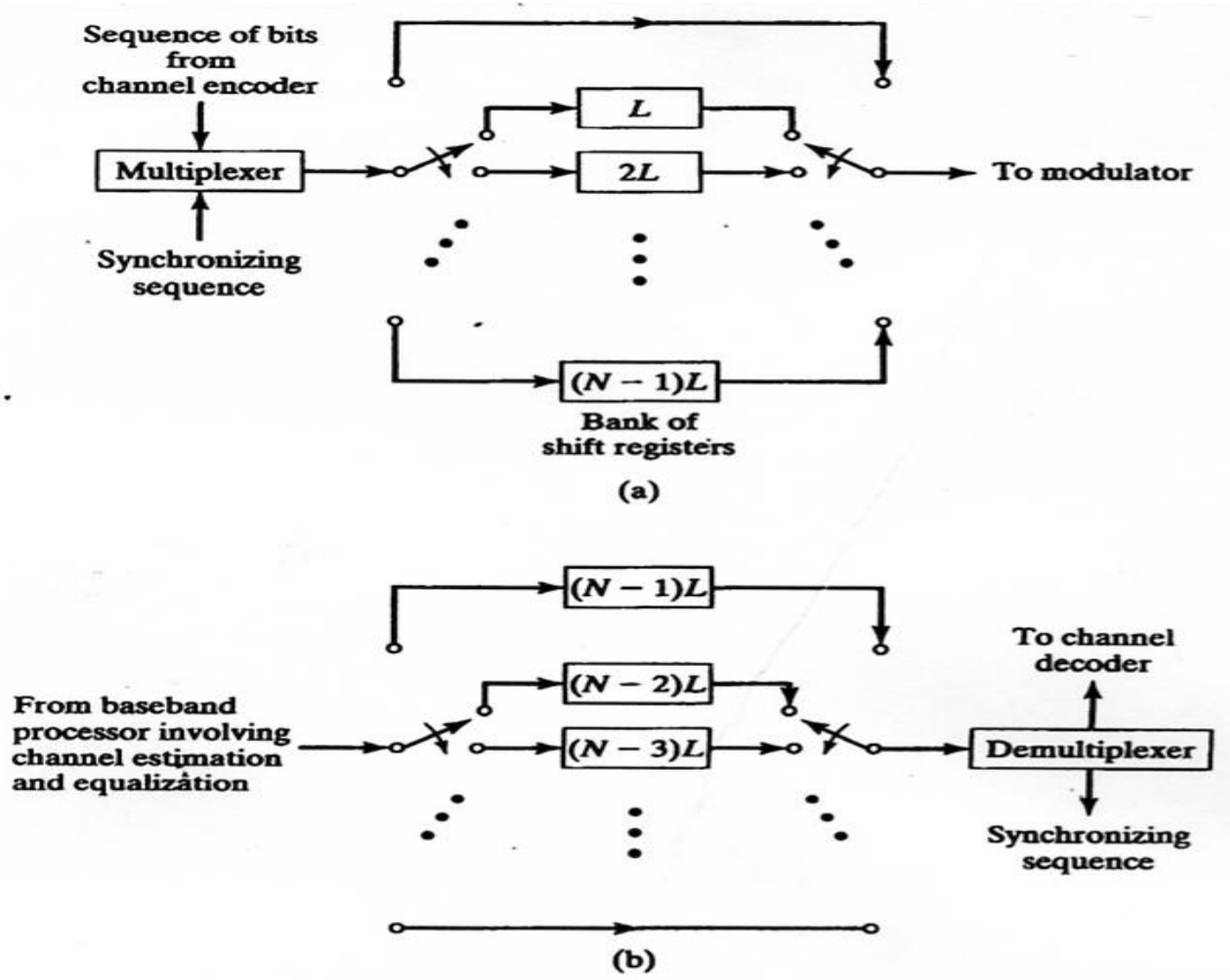
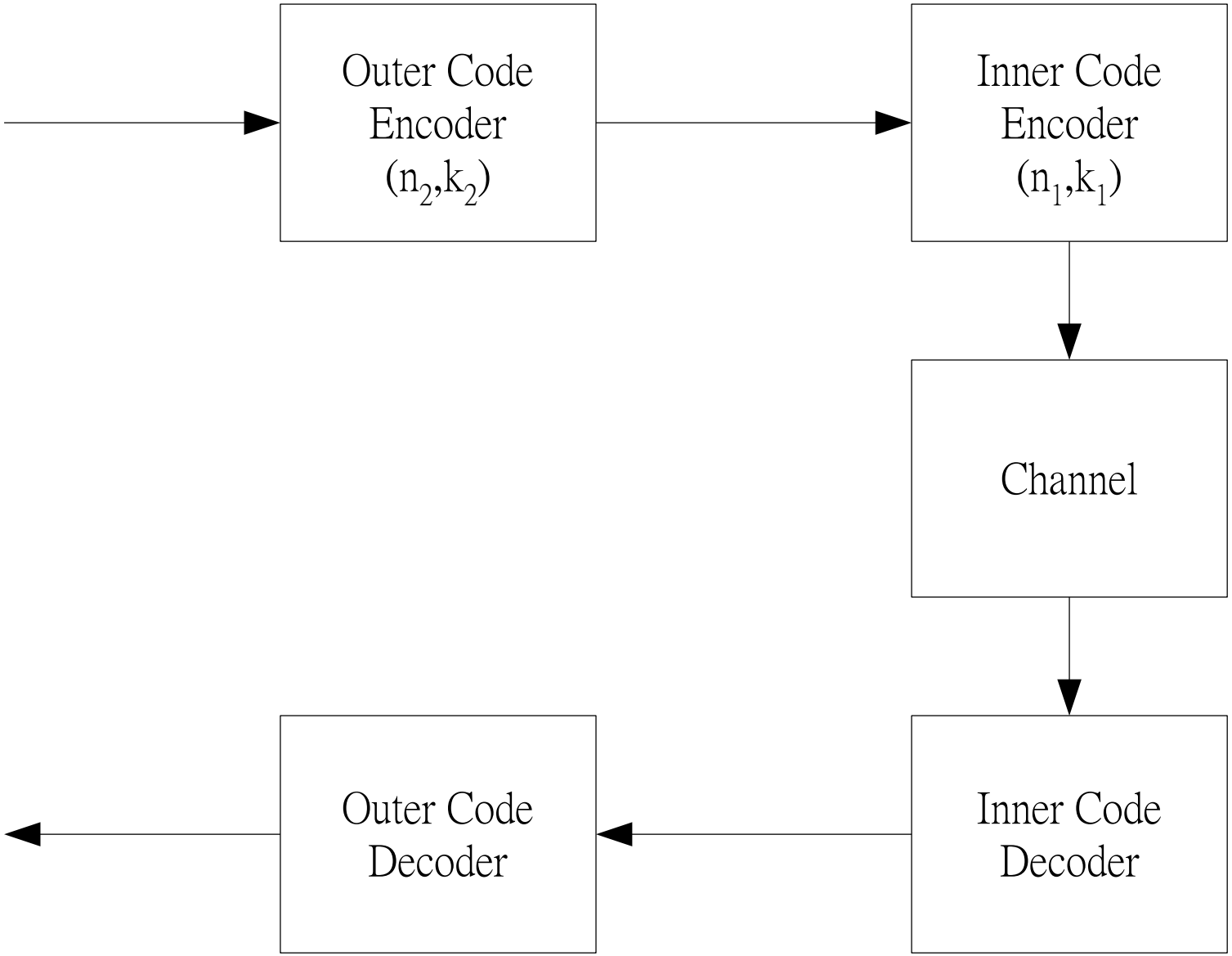


FIGURE 4.12 (a) Convolutional interleaver. (b) Convolutional deinterleaver.

7.5.3 Concatenated Coding Scheme

- Concatenation is a very effective method of constructing long powerful codes from shorter codes.
- It was devised by Forney in 1965.

  and is often used to achieve high reliability with reduced decoding complexity.
- A simple concatenated code is formed from two codes：an (n_1 ,k_1) binary code C_1 and an (n_2,k_2) nonbinary code C_2 with symbols from GF(), say a RS code.
- Concatenated codes are effective against a mixture of random errors and burst errors. Scattered random errors are corrected by C_1. Bursts may affect relatively few bytes, but probably so badly that C_1 cannot correct them. These few bytes can then be corrected by C_2.


Outer Code Encoder (n_2,k_2)
Inner Code Encoder (n_1,k_1)
Channel
Inner Code Decoder
Outer Code Decoder

Encoding

- Encoding consists of two stages, the outer code encoding and the inner code encoding, as shown in below.
- First a message of k_1k_2 bits are divided into k_2 bytes of k_1 bits each. Each *k1*-bit byte is regarded as a symbol in GF( ).

  This k_2-byte message is encoded into an n_2-byte codeword in C_2.
- Each k_1-bit byte of is then encoded into an n_1-bit codeword in C_2.
- This results in a string of n_2 codewords in C_2, a total of n_1n_2 bits.
- There are a total of such strings which form an (n_1n_2,k_1k_2) binary linear code, called a concatenated code.
- C_1 is called the inner code and C_2 is called the outer code.
- If the minimum distances of the inner and outer codes are d_1 and d_2 respectively, the minimum distance of their concatenation is at least $d_1 d_2$.

Decoding

- Decoding of a concatenated code also consists of two stages, the inner code decoding and the outer code decoding, as shown in the above figure.
- First, decoding is done for each inner codeword as it arrives, and the parity bits are removed. After n_2 inner codewords have been decoded, we obtain a sequence of n_2 k_1-bit bytes.
- This sequence of n_2 bytes is then decoded based on the outer code C_2 to give $k_1 k_2$ decoded message bits.
- Decoding implementation is the straightforward combination of the implementations for the inner and outer codes.

Error Correction Capability

- Concatenated codes are effective against a mixture of random errors and bursts.
- In general, the inner code is a random-error-correcting code and the outer code is a RS code.
- Scattered random errors are corrected by the inner code, and bursts are then corrected by the outer code.
- Various forms of concatenated coding scheme are being used or proposed for error control in data communications, especially in space and satellite communications.
- In many applications, concatenated coding offers a way of obtaining the best of two worlds, performance and complexity.

References

1.J.B. Cain, G.C. Clark, Jr. , and J.M. Geist , " Punctured Convolutional Codes of Rate n-1 / n and Simplified Maximum Likelihood Decoding , " IEEE Trans. Inform. Theory, Vol.25 , No.1 , pp. 97-100 , Jan.1979.

2. J. Hagenauer " Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications , IEEE Trans. Commun., Vol.36 , No. 4 , pp.387-400 , Apr. 1988.

Homework : Chapter 7

1. Consider a (3,1,3) convolutional code with generator plolynomials :

$g^{(1)} = 1+ D + D^2 + D^3$, $g^{(2)} = 1+ D + D^3$,

$g^{(3)} = 1+ D^2 + D^3$

a. Draw the encoder block diagram

b. Draw the state diagram

c . Draw the trellis diagram of this code with a message sequence of length L= 5

2. Consider a (2,1,3) convolutional code with

$g^{(1)} = 1+ D^2$, $g^{(2)} = 1+ D + D^2 + D^3$

a. Draw the block diagram of the encoder

b. Draw the state diagram

c. Draw the trellis diagram for information length L=4

d. For a BSC , $p < 0.5$, using Viterbi algorithm to decode the received sequence r = 10,01, 00,00,11, 11, 00

Fin the codeword transmitted and the corresponding message bits.