ECE 8700: Communication Systems Engineering

Module 6
Forward Error Correction

Mojtaba Vaezi
http://www.ece.villanova.edu/ mvaezi

Villanova University

May 1, 2025

Contents

1 Introduction **4**

1.1 Modulation and Coding 6

2 Channel Coding **9**

2.1 Why Use Channel Coding? 10

2.2 Code Rate and Redundancy 12

2.3 Parity-Check Codes 13

2.3.1 Single-Parity Code 13

2.3.2 Rectangular-Parity Code 14

2.4 Linear Block Codes 18

2.5 The Hamming Codes 24

2.5.1 Overview of the Hamming (7,4) Code 25

2.6 Binary Linear Convolutional Codes 36

2.6.1 Historical Background of Convolutional Codes 36

2.6.2 Structure of a Convolutional Encoder 39

2.6.3 Viterbi Decoding Algorithm 55

2.6.4 Convolutional Code Decoding Details 65

2.6.5 Soft-decision (Soft-input) Decoding* 71

2.7 Transfer Function & Weight Distribution Polynomial 74

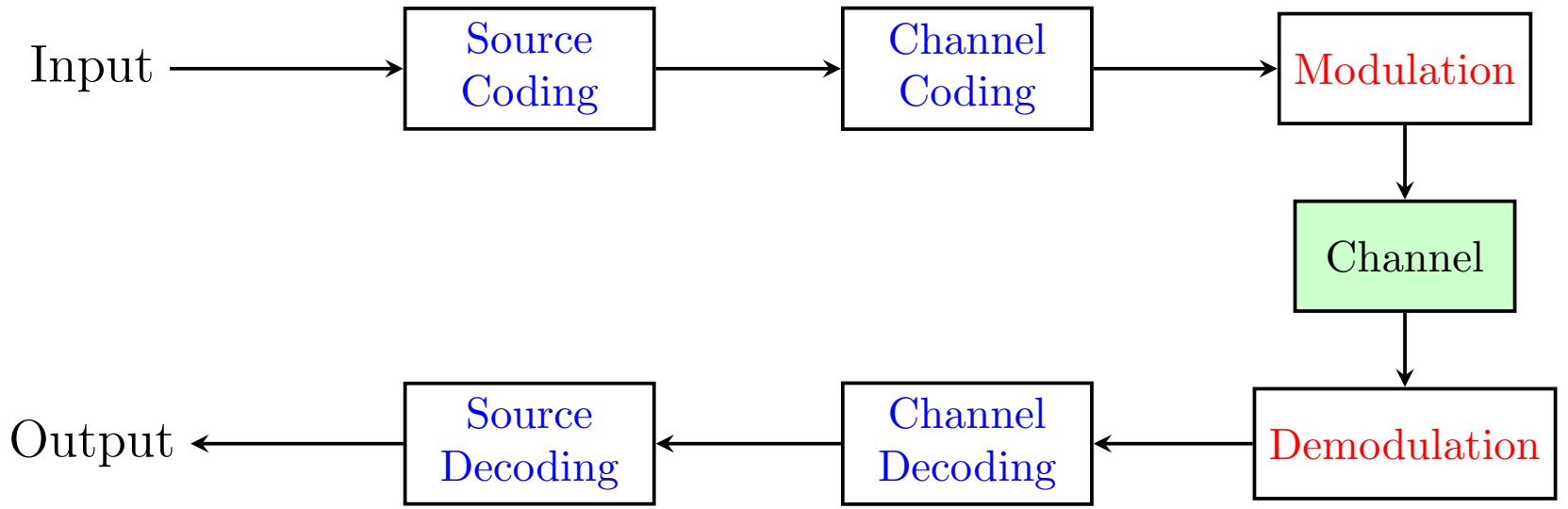
2.7.1 Weight Distribution Polynomial for Linear Block Codes 74
2.7.2 Transfer Function of Convolutional Codes 75

3 Trellis Coded Modulation (TCM)* 82
3.1 Set Partitioning 86

4 Summary 95

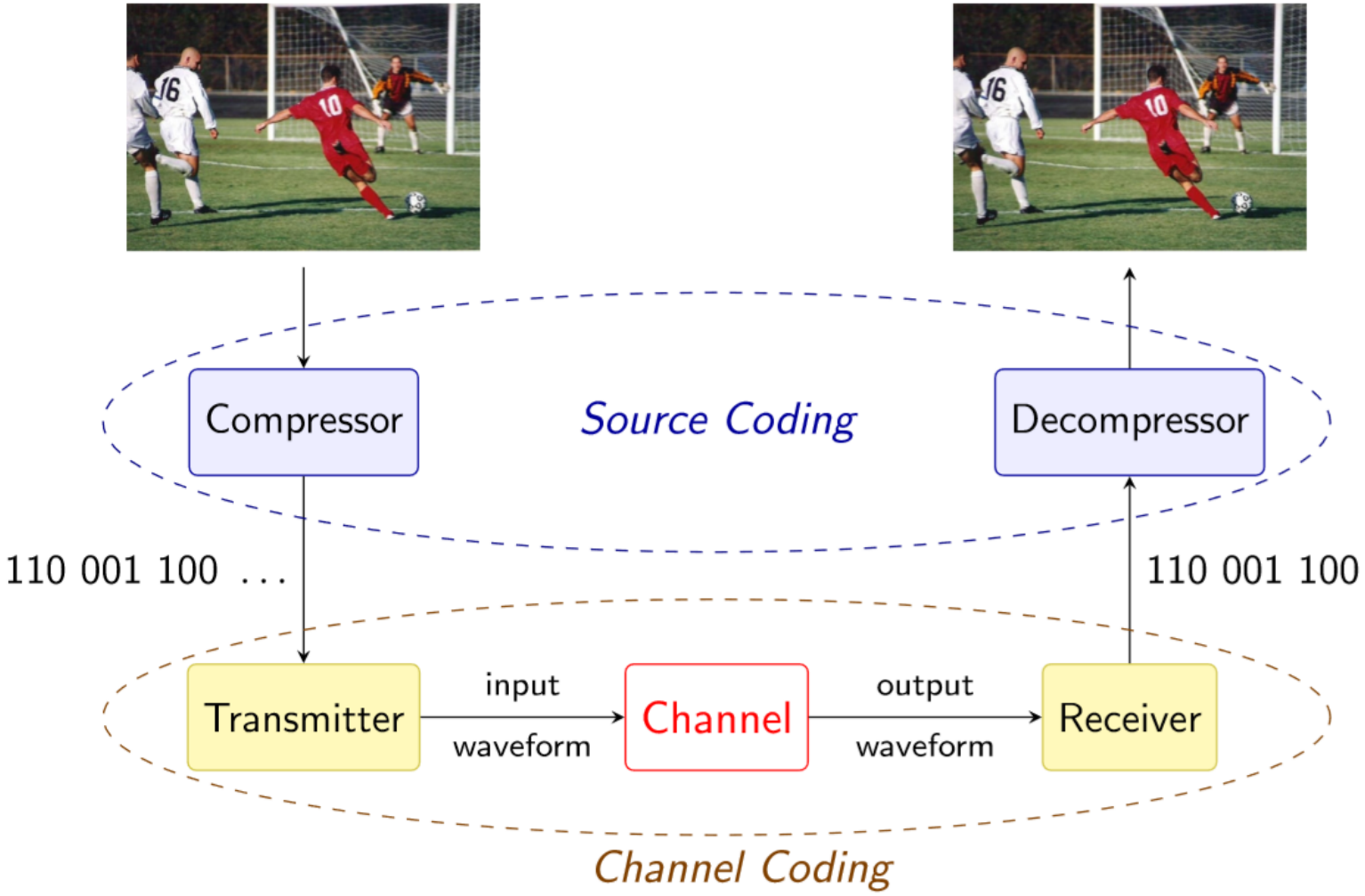
1 Introduction

- Recall that a digital transmission scheme generally involves



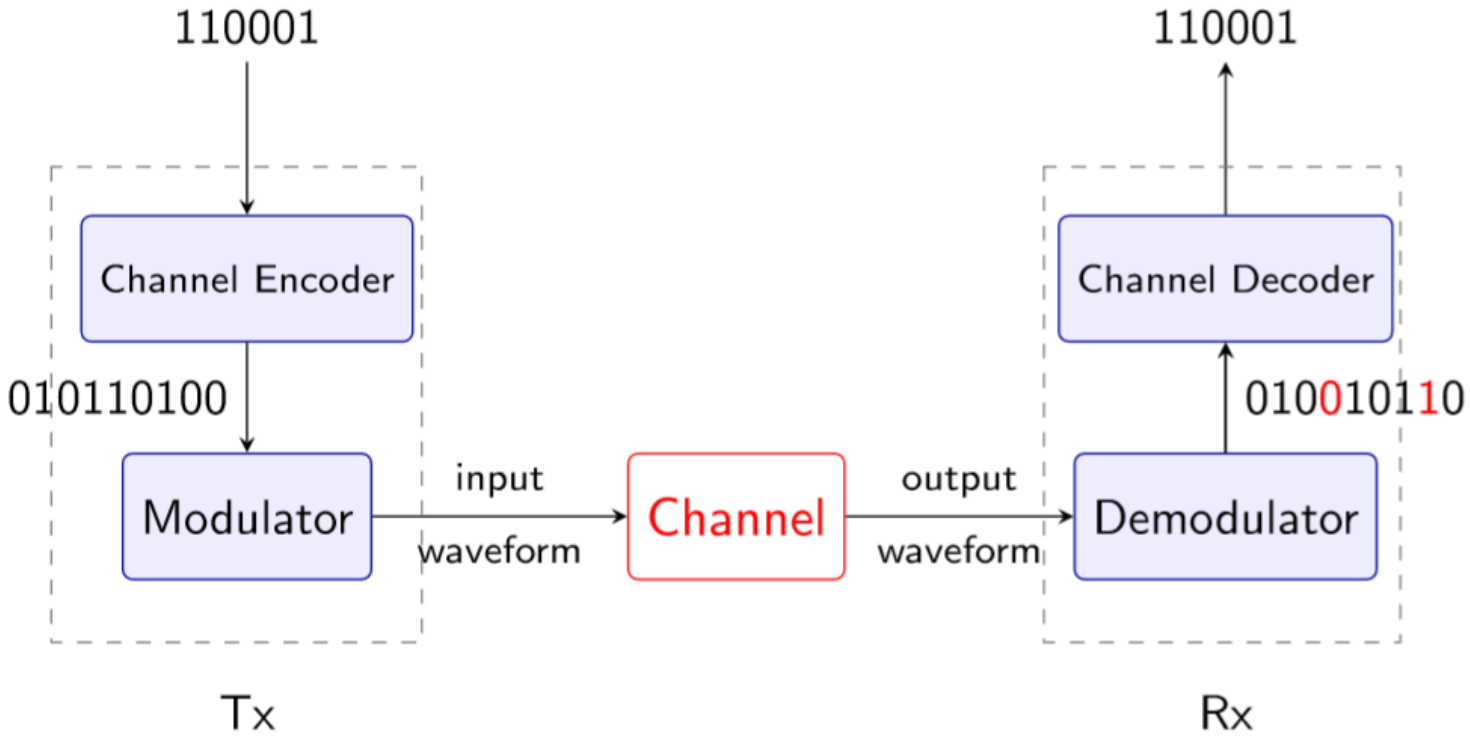
- Input/output are considered digital (if analogue sampled/quantized)
- Your cell-phone, for example, contains a pair of transmitter and receiver, consisting of a **CODEC** and **MODEM**

Channel Coding



- The channel coding part is often modeled as a binary symmetric channel (BSC)

1.1 Modulation and Coding



- Joint coding-modulation schemes (e.g., TCM, BICM) offer optimal performance.
- Separation is common due to flexibility, legacy systems, and cost constraints.

Design Considerations for Coding and Modulation

- Joint Optimization
  - Coding and modulation should ideally be **co-designed** to
    - Maximizes spectral efficiency/lower BER
  - **Examples:**
    - `Trellis-Coded Modulation (TCM)`: Integrates convolutional codes with modulation
    - `Bit-Interleaved Coded Modulation (BICM)`: Combines modern codes (LDPC, Turbo) with higher-order QAM
- Separate Design
  - Common in practice due to
    - **Modularity**: Independent upgrades (e.g., new codes with existing modulators)
    - **Standardization**: Fixed modulation schemes (QPSK, 16-QAM) with generic codes
    - **Complexity**: Joint designs has a higher complexity

Standards Implementing BICM

- **3GPP LTE (Long Term Evolution)**
  - BICM is used in both uplink and downlink for robust channel coding with QAM modulation.
  - BICM framework: Turbo coding + bit-level interleaving + QAM.
- **5G NR (New Radio)**
  - BICM is part of the physical layer with LDPC coding (for data) and Polar coding (for control), followed by interleaving and higher-order modulations (e.g., 256-QAM).
  - BICM-like structure with flexible modulation and coding schemes (MCS).
- **IEEE 802.11 (Wi-Fi)**
  - Standards like 802.11a/g/n/ac use BICM with convolutional coding and interleaving before modulation (e.g., 64-QAM).
  - Helps mitigate burst errors due to multipath fading.
- **DVB (Digital Video Broadcasting)** and **WiMAX (IEEE 802.16)**

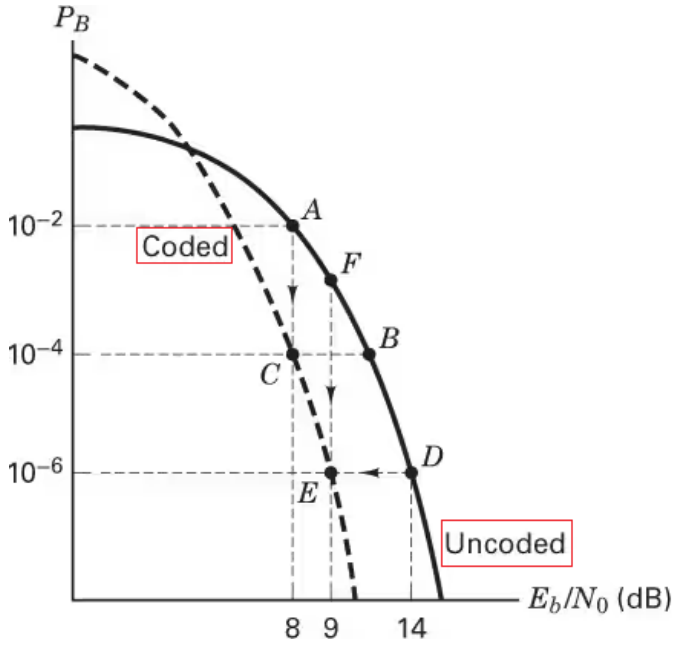
2 Channel Coding

Source coding removes **redundancy**, whereas **channel coding** adds structured redundancy for error control.

- Two main methods use structured redundancy for error control
  - **Error detection and retransmission:** Uses parity bits to detect errors. Receiver requests retransmission. Requires a two-way link.
  - **Forward error correction (FEC):** Uses parity bits for both detection and correction. Requires only a one-way link.
- Not all error patterns can be corrected; codes are classified by their error-correcting capability. Some well-known examples od channels codes are
  - **Hamming codes** (1950)
  - BCH and Reed–Solomon codes (1960)
  - **Convolutional codes** (1955)
  - Turbo codes (1993) and LDPC codes (1962, rediscovered in 1996)
  - Polar codes (2008)

2.1 Why Use Channel Coding?

- **Coded vs. encoded performance.**
  - **Improved BER** at same SNR (e.g., A → C).
  - **Lower SNR** for same BER (e.g., B → C).
  - **Better performance ceiling** (e.g., D → E).



- **Coding gain** SNR reduction (in dB) for same BER using coding.
- $G = \left.\frac{E_b}{N_0}\right|_{\text{uncoded}} - \left.\frac{E_b}{N_0}\right|_{\text{coded}}$

Channel Coding

- Coding trade-offs
  - **Trade-Off 1:** Coding improves error performance at the cost of increased bandwidth.
  - **Trade-Off 2:** Coding allows reduced power (or E_b/N_0) while maintaining error performance, at the cost of bandwidth.
  - **Trade-Off 3:** Coding enables higher data rates without sacrificing error performance, again at the cost of more bandwidth.
  - **Trade-Off 4:** Coding increases user capacity by lowering required power per user, with expanded bandwidth as the trade-off, e.g.,
    - In code division multiple access (CDMA), lower power $\rightarrow$ lower interference $\rightarrow$ higher users and system capacity
    - In general, enables getting closer to Shannon capacity limit

2.2 Code Rate and Redundancy

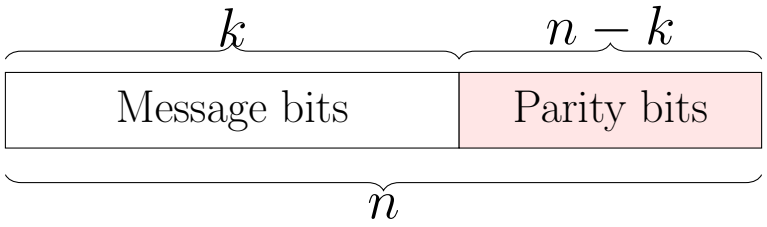
– Source data is divided into k**-bit** blocks (**message** bits).

– Each block is encoded into n **bits** (code bits, or **codewords**).

– $(n-k)$ **bits** are redundant (**parity**/check bits).

– Code is called **an** (n,k) **code**.

An (n,k) Block Codes



– Redundancy: ratio of redundant bits to data bits.

– **Code rate:** k/n (info per code bit).

– Lower rate $\Rightarrow$ more redundancy and higher bandwidth.

– **Example:** rate 1/2 doubles bandwidth; rate 3/4 adds 33% overhead.

2.3 Parity-Check Codes

2.3.1 Single-Parity Code

- A basic error-detection code that adds a **single** parity bit to a binary message.
- The parity bit ensures the total number of 1s in the codeword is even or odd.
- Two common types: even parity and odd parity
  - **Even parity:** Total number of 1s (including parity bit) is even.

Example 1. *Even Parity.*

Message	Parity	Codeword
000	0	0000
100	1	1100
010	1	1010
110	0	0110
001	1	1001
101	0	0101
011	0	0011
111	1	1111

- Error Detection Capability
  - They detect all single-bit errors (in fact, can detect all odd number of errors!)
  - Error is detected, but the position of the error is unknown.
  - Thus, cannot correct any errors.
- **Example**
  - If one bit is flipped: 1**0**11 → parity check fails (odd)
- Widely used in memory systems and communication protocols.

2.3.2 Rectangular-Parity Code

Idea: start with rectangular array of message bits, add parity checks for each row and column.

m_1	m_2	p_1
m_3	m_4	p_2
p_3	p_4	

Q: What are is n and k?

Example 2. *Consider the three rectangular parity codewords (possibly erroneous) below. Determine the possible error position(s) in each.*

0 1 1	0 1 1	0 1 1
1 1 0	1 0 0	1 1 1
1 0	1 0	1 0

Solution:

0 1 1	0 1 1	0 1 1
1 1 0	1 0 0	1 1 1
1 0	1 0	1 0

- Possible conclusions
  - **Codeword 1:** Parity for each row and column is correct ⇒ no errors
  - **Codeword 2:** Parity check fails for row #2 and column #2 ⇒ m_4 is incorrect
  - **Codeword 3:** Parity check only fails for row #2 ⇒ bit p_2 is incorrect

Hamming Distance

- Richard W. Hamming (1915-1998) was an American mathematician known for foundational contributions to computer engineering and telecommunications, including the
  - **Hamming code**,
  - Hamming distance,
  - Hamming window, and
  - Sphere-packing (Hamming bound) [photo: Wiki]



- The **Hamming distance** between two binary strings (of equal length) is the number of positions where two bit strings differ.

  Example: $d_H\{000000,\ 110010\} = 3$

Hamming distance is used to measure how many bits must be changed to convert one codeword into another.

- Denoted mathematically as

$$d_H(x, y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

- Hamming distance is fundamental in error detection and correction.
- A code with minimum Hamming distance d
  - can **detect all** patterns of up to $d - 1$ bit errors.
  - can **correct all** patterns of up to $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ bit errors.

2.4 Linear Block Codes

A subset S of a **binary** vector space V that is called a `subspace` if

1. The **all-zero vector** is in S
2. $u, v \in S \implies u + v \in S$ (closed under addition)

The above properties are fundamental to linear block codes.

Definition 1. ***[Linear block code]*** *A binary (n, k) block code $\mathcal{C}$ is linear* ***if and only if*** *the sum (XOR) of any two codewords is also a codeword. That is,*

$$\mathbf{c}_i \oplus \mathbf{c}_j \in \mathcal{C}, \quad \textit{for all } \mathbf{c}_i, \mathbf{c}_j \in \mathcal{C}$$

This implies,

- *all-zero codeword must be a valid codeword of any linear code*
- *vectors outside the subspace cannot be created by adding two legitimate codewords.*

- An (n, k) linear code has a code rate of $\frac{k}{n}$.
- Sometimes written as (n, k, d), where d is the minimum Hamming distance.
- The **weight** of a codeword ($w_H(\mathbf{c}_i)$) is the number of 1's it contains.
- For a linear code, the **minimum Hamming distance** d is equal to the smallest number of 1s (**minimum weight**) in any nonzero codeword.

Example 3. *Which of the following codes are linear? For the linear codes, determine n, k, and d.*

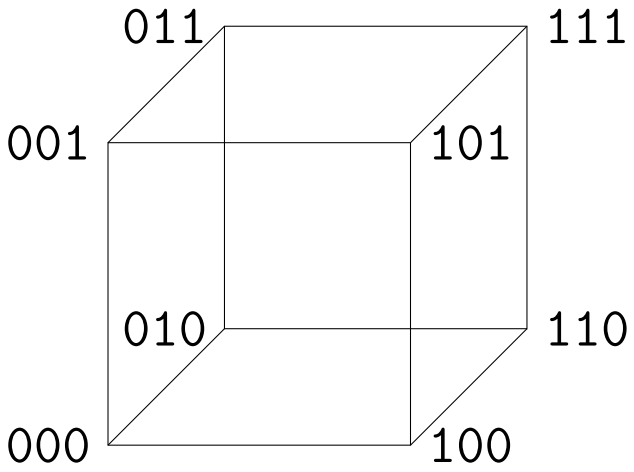
- $\{000,\ 111\}$ — *linear code:* $(n, k, d) = (3, 1, 3)$
- $\{0000,\ 1100,\ 0011,\ 1111\}$ — *linear code:* $(n, k, d) = (4, 2, 2)$
- $\{1111, 0000, 0001\}$ — *non-linear code!*
- $\{1111, 0000, 0010, 1100\}$ — *non-linear code!*

How to Construct Codes?

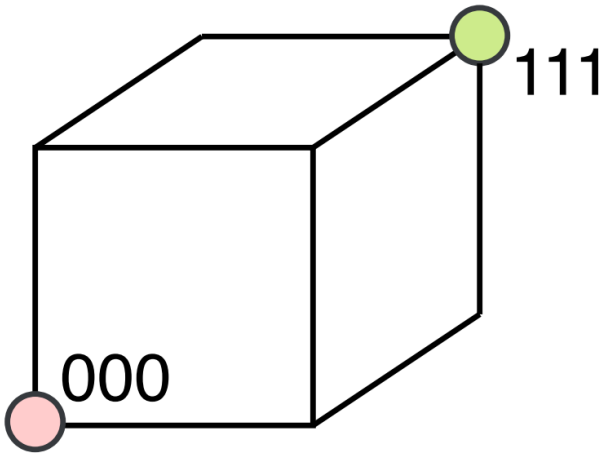
Idea: Embedding for Structural Separation

- Encode so that codewords are far apart from each other.
- Ensure likely error patterns do not transform one codeword into another.

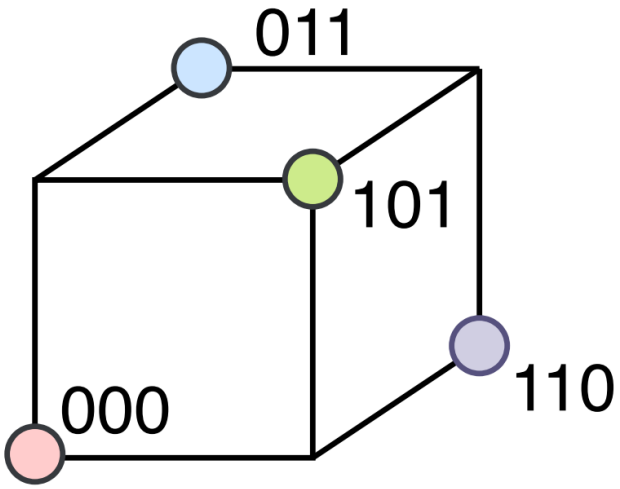
$n = 3$



- **Code:** nodes chosen in **hypercube** + mapping of message bits to nodes.
  - If we choose 2^k out of 2^n nodes, it means we can map all k-bit message strings into a space of n-bit codewords. The code rate is k/n.

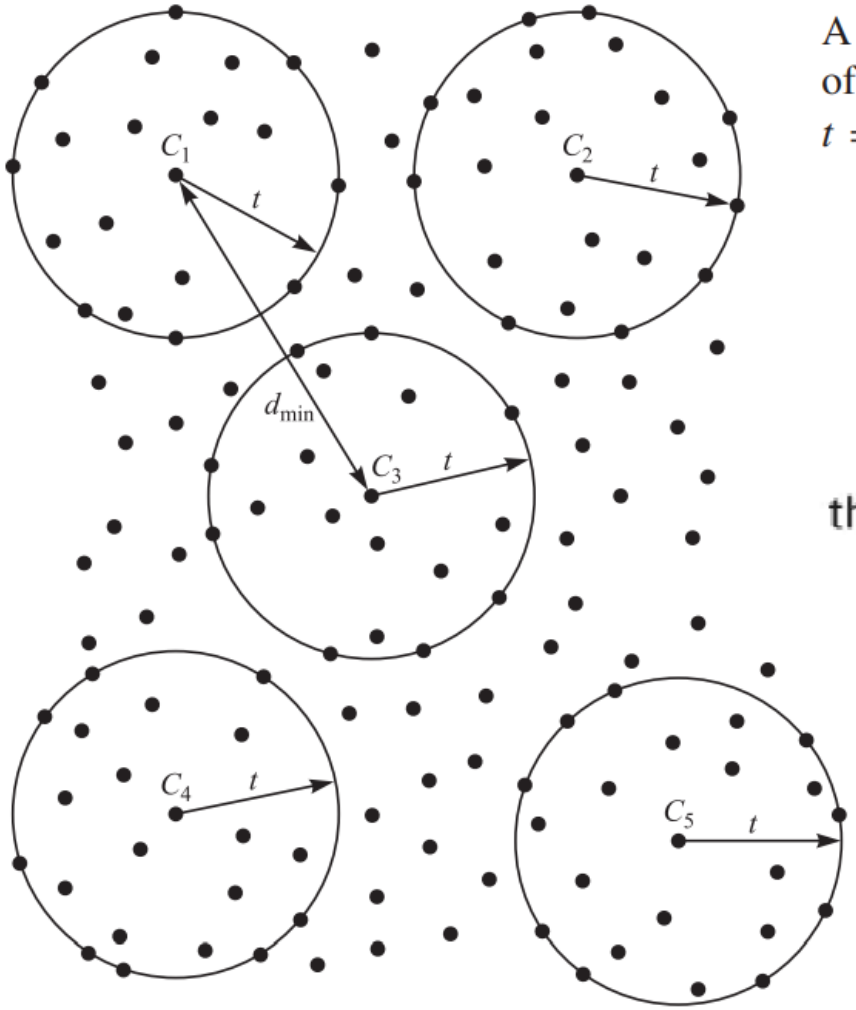


$(n, k, d) = (3, 1, 3)$, rate $= \frac{1}{3}$



$(n, k, d) = (3, 2, 2)$, rate $= \frac{1}{2}$

- **Code 1:** with 2 codewords $\{000, 111\}$. This code has a distance of 3. In this case, any single or double-bit error can be detected. If double-bit errors are not likely to happen, then this code can correct any single-bit error along with any single-bit error detection. This is a repetition code.
- **Code 2:** with 4 codewords $\{000, 011, 101, 110\}$. This code has a distance of 2, as any pair of code words differ by 2 bits from each other. Since the distance is 2, it can detect any single-bit error.



A representation of codewords as center of spheres with radius $t = \left\lfloor \frac{1}{2}(d_{\min} - 1) \right\rfloor$.

2^n n-tuples constitute the entire space

2^k n-tuples constitute the subspace of codewords

Designing a Code for 4-bit Messages with Single-Error Correction

Example 4. *Construct a code for 4-bit messages that allows single-error correction.*

Solution:

- 4-bit messages $\rightarrow$ 16 messages and 16 codewords
- Single-error correction $\rightarrow$ *minimum Hamming distance* $d_{\min} = 3$
- We can check that with 5-bit or 6-bit codewords it is not possible to have $d_{\min} = 3$.
- Thus, we need at least 7-bit codewords

$$
\begin{array}{cccc}
0000000 & 1100001 & 1100110 & 0000111 \\
0101010 & 1001011 & 1001100 & 0101101 \\
1010010 & 0110011 & 0110100 & 1010101 \\
1111000 & 0011001 & 0011110 & 1111111
\end{array}
$$

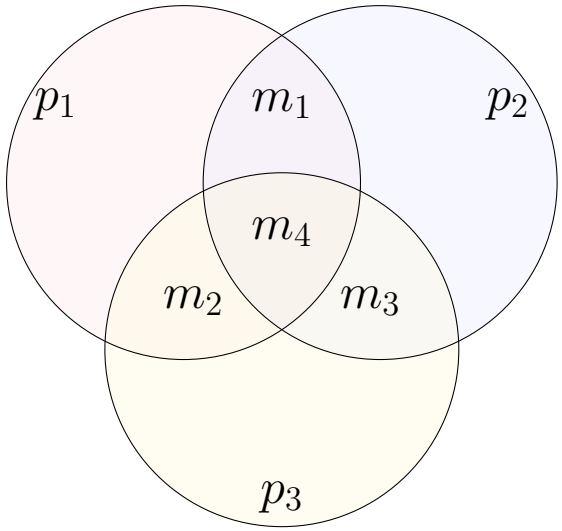
2.5 The Hamming Codes

- **Hamming codes** are a class of linear error-correcting codes.
- They can **detect up to two-bit errors** and **correct one-bit errors** (minimum Hamming distance is 3).
- Designed for binary messages with the form:

$$(n, k) = (2^\ell - 1, 2^\ell - 1 - \ell), \qquad \ell = 2, 3, \ldots$$

  where ℓ is the number of parity bits.
- Parity bits are placed at positions that are powers of 2 (e.g., 1, 2, 4, 8, ...).
- **Common example:** (7,4) Hamming code — encodes 4 data bits into 7 bits by adding 3 parity bits.

2.5.1 Overview of the Hamming (7,4) Code



***Modulo-2* addition**, aka **XOR**

$$p_1 = m_1 \oplus m_2 \oplus m_4$$
$$p_2 = m_1 \oplus m_3 \oplus m_4$$
$$p_3 = m_2 \oplus m_3 \oplus m_4$$

- **Code parameters:** $n = 7$, $k = 4$ (4 information bits, 3 parity bits).
  - **Message:** a row vector $\mathbf{m}$ of length k
  - **Codeword:** a row vector $\mathbf{c}$ of length n
  - **Parity bits:** a row vector $\mathbf{p}$ of length $n - k$
- **Generator matrix:** The systematic generator matrix G for a linear (n, k) block code is

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

$G = [I_4 \,|\, P]$, where

$$P = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad G = \left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

- **Codeword**: $\boxed{\mathbf{c} = \mathbf{m}G}$
- **Example:** For $\mathbf{m} = \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix}$, we have

$$\mathbf{c} = \mathbf{m}G = \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 3 & 2 \end{pmatrix} \overset{\text{mod } 2}{\equiv} \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

- **Minimum Hamming distance:** $d = 3 \Rightarrow t = 1$ (1 error is correctable)

Parity-Check Matrix

For a code with $G = [I_k \mid P]$, the parity-check matrix H will be of size $(n-k) \times n$, and it takes the form

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

- The corresponding parity-check matrix for the $(7, 4)$ Hamming code is

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

  - **Syndrome** is obtained as $\boxed{\mathbf{s} = \mathbf{r}H^T}$
  - Note that in general $\mathbf{r} = \mathbf{c} + \mathbf{e}$, and we have

$$\mathbf{s} = \mathbf{r}H^T = (\mathbf{c} + \mathbf{e})H^T = \mathbf{c}H^T + \mathbf{e}H^T = \mathbf{m}GH^T + \mathbf{e}H^T = \mathbf{0} + \mathbf{e}H^T$$

    This is because $GH^T = 0$
  - Thus, $\mathbf{s}$ is the syndrome of error; as $\mathbf{s} = \mathbf{e}H^T$.

- In light of the above, we can create a syndrome table to determine the location of error.
- For example, if $\mathbf{e} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$, we have

$$\mathbf{s} = \mathbf{e}H^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

Table 1: Syndrome Look-Up Table

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

Error Detection and Correction

Example 5. *Assume the transmitted codeword, using the Hamming* $(7, 4)$ *code, is*

$$\mathbf{c} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Find the syndrome for the received vector $\mathbf{r}$ *in each of the following cases:*

- *No errors occur during transmission (i.e.,* $\mathbf{r} = \mathbf{c}$*).*
- *The last bit is flipped during transmission.*

- If $\mathbf{r} = \mathbf{c}$, then

$$\mathbf{s} = \mathbf{r}H^T = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{pmatrix} 2 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \Longrightarrow \text{ no errors}$$

- If $\mathbf{r} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$, then

$$\mathbf{s} = \mathbf{r}H^T = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{pmatrix} 2 & 4 & 3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \neq \mathbf{0} \implies \text{there is an error}$$

- Using the syndrome table, we can tell the location of the error as well.
- Equivalently, since $\mathbf{e} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$, we have

$$\mathbf{s} = \mathbf{e}H^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

- More examples and details
  - **Transmitted codeword:** $\mathbf{c} = 0101100$
  - **No error:** Received vector $\mathbf{r} = 0101100$, syndrome $\mathbf{s} = \mathbf{r}H^T = 000$
  - **1-bit error** at position 5

$$\mathbf{r} = 0111100 \Rightarrow \mathbf{s} = 100 \overset{\text{Synd.Table}}{\Longrightarrow} \text{error in column 5}$$

  - **2-bit error** in positions 1 and 7

$$\mathbf{r} = 1101101 \Rightarrow \mathbf{s} = 111 \Rightarrow \text{wrong correction, but error detected}$$

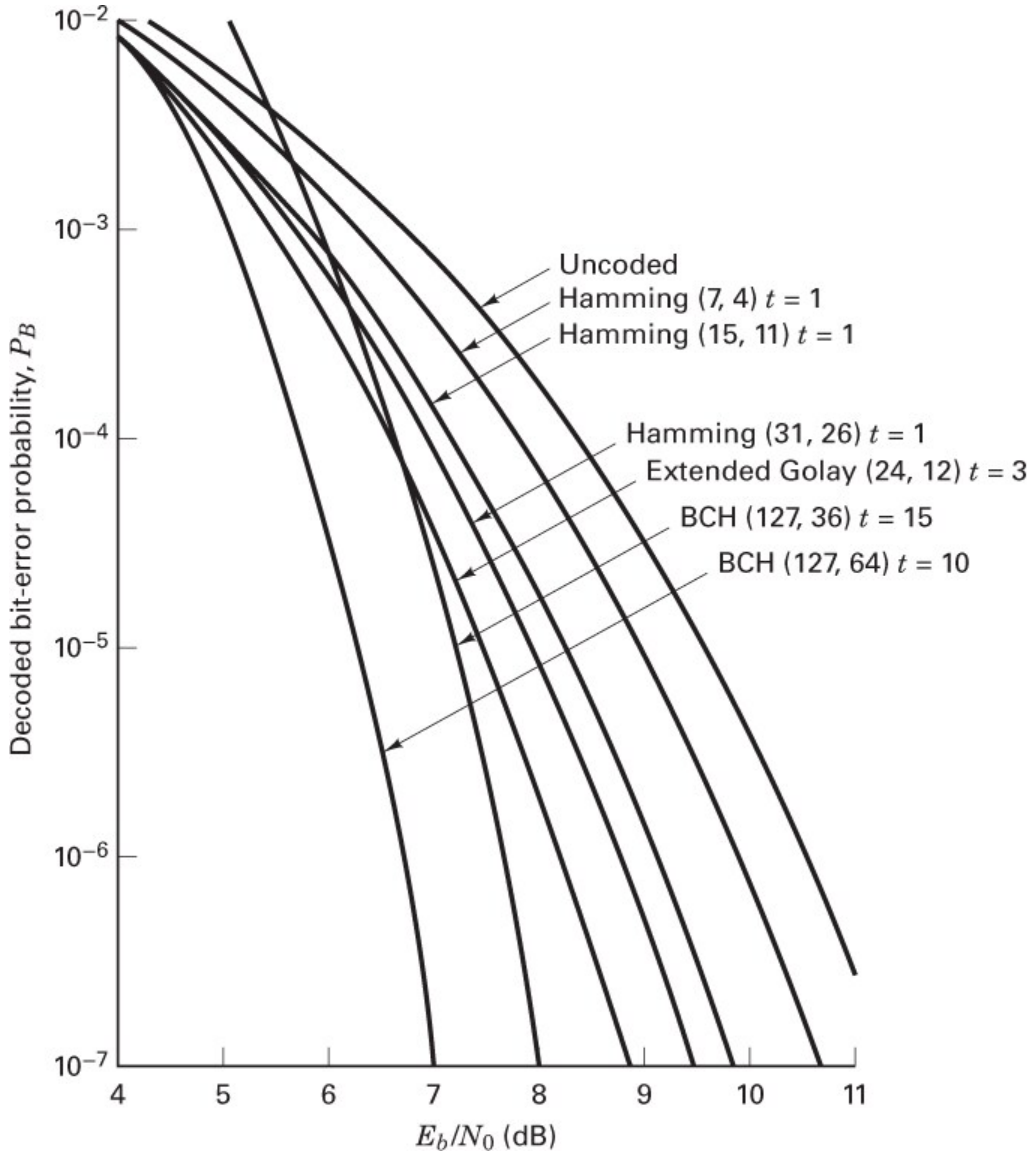
  - **4-bit error** in positions 1, 3, 4, 5

$$\mathbf{r} = 1110000 \Rightarrow \mathbf{s} = 000 \Rightarrow \text{undetected error}$$

  - **5-bit error** in positions 1, 3, 5, 6, 7

$$\mathbf{r} = 1111011 \Rightarrow \mathbf{s} = 100 \Rightarrow \text{error detected (good)}$$

- Number of **guaranteed correctable errors**: 1
- Number of **guaranteed detectable errors**: 2
- Some patterns with more than 2 errors are still detectable, but not always.

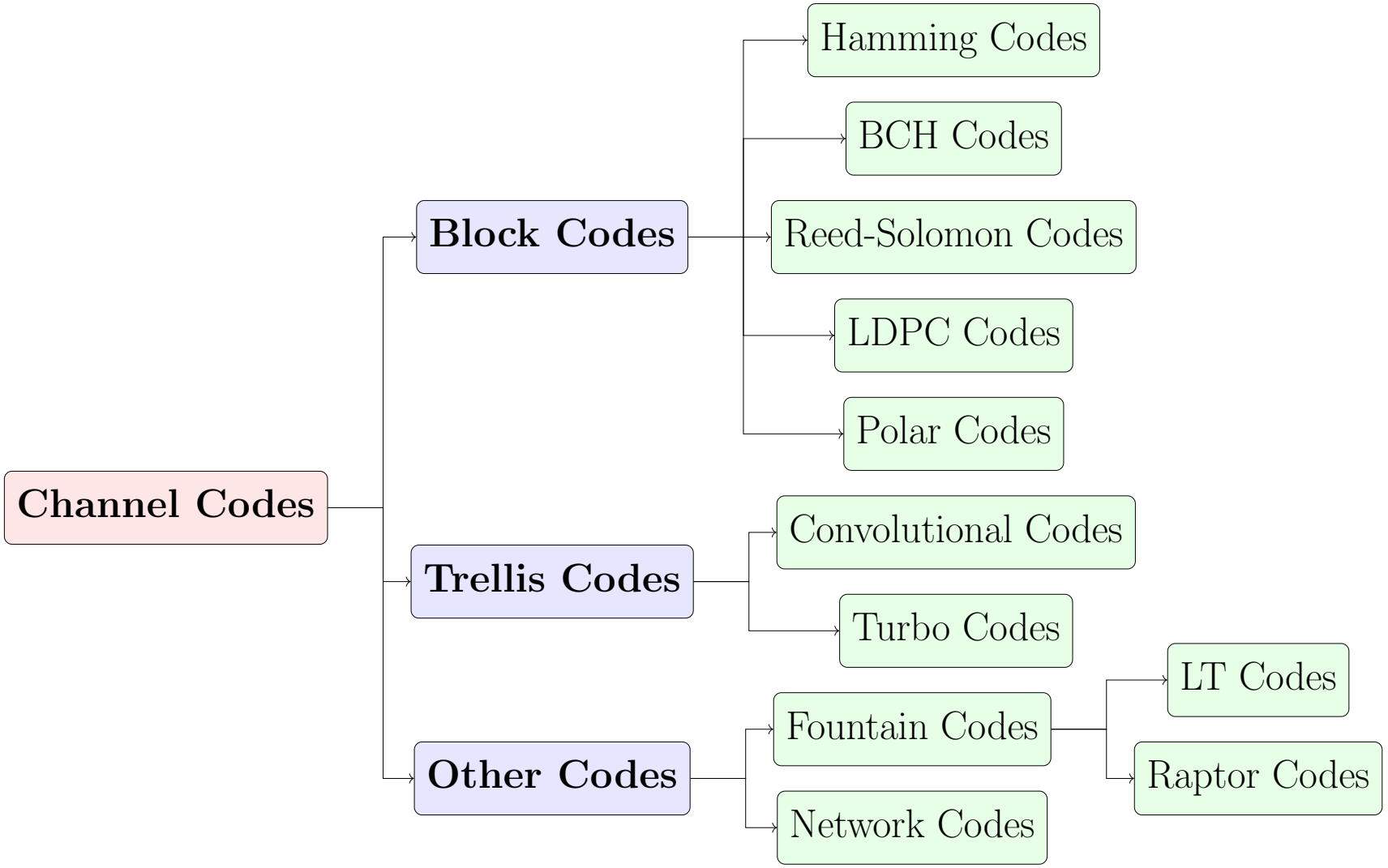

Decoded bit-error probability, P_B
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
4
5
6
7
8
9
10
11
E_b/N_0 (dB)
Uncoded
Hamming (7, 4) t = 1
Hamming (15, 11) t = 1
Hamming (31, 26) t = 1
Extended Golay (24, 12) t = 3
BCH (127, 36) t = 15
BCH (127, 64) t = 10

Linear Block Codes Summary

Key parameters in any (n, k, d) linear block codes (not just the Hamming code)

- Key components
  - Message vector: $\mathbf{m}$
  - Generator matrix: G
  - Parity-check matrix: H
- Codeword: $\mathbf{c} = \mathbf{m}G$
- Syndrome: $\mathbf{s} = \mathbf{r}H^T$ (for every valid codeword, the syndrome is zero since $GH^T = 0 \implies \mathbf{s} = 0$)
- Rate and error capabilities
  - code rate is k/n.
  - can **detect** **all** patterns of up to $d - 1$ bit errors.
  - can **correct** **all** patterns of up to $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ bit errors.

Channel Coding



Block Codes vs. Trellis Codes

- **Block Code**
  - Data stream is divided into blocks of k symbols (datawords).
  - Each dataword is encoded into a block of n symbols (codeword).
  - Codewords are **independent** and concatenated to form the code stream.
  - **Example:** Hamming codes
- **Trellis Code**
  - Data stream divided into blocks of k symbols (dataframes).
  - Each dataframe is encoded into an n-symbol codeframe.
  - Codeframes are **dependent** on multiple dataframes.
  - **Example:** Convolutional codes

2.6 Binary Linear Convolutional Codes

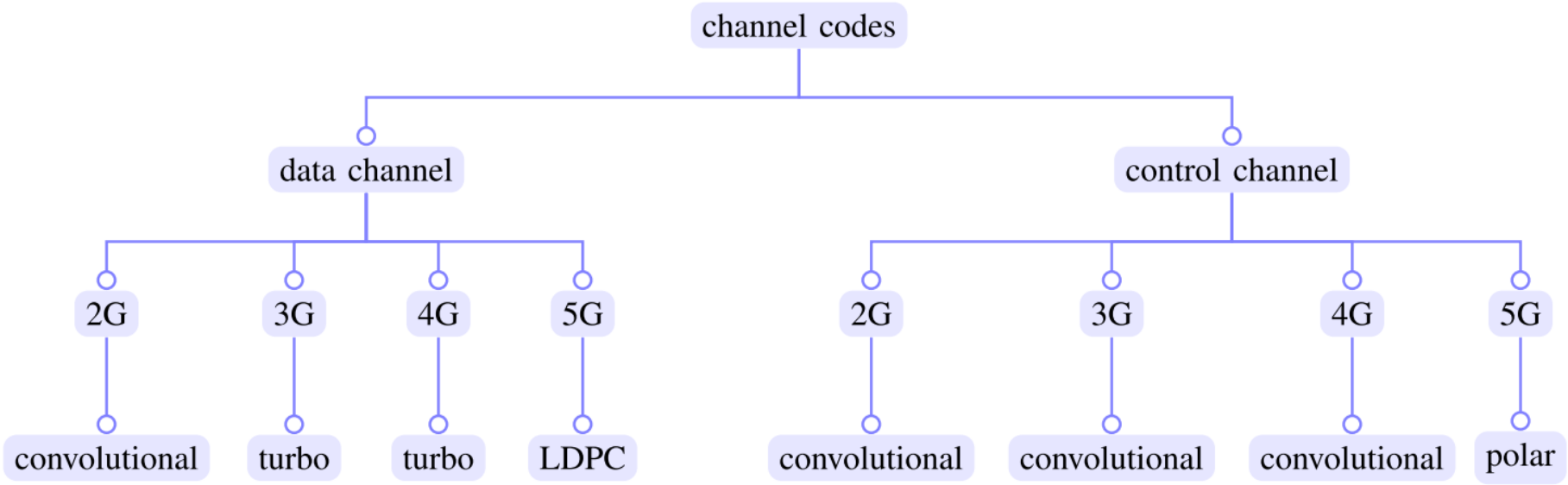
2.6.1 Historical Background of Convolutional Codes

1955	**Elias** introduces convolutional codes as alternative to block codes
1957	**Wozencraft** proposes **sequential decoding** algorithm
1967	**Viterbi** introduces **Viterbi algorithm** for decoding (relatively easy to implement with small memory orders)
1968	**Omura** shows equivalence of the the Viterbi algorithm to finding the **shortest path** in weighted graph
1974	**Forney** recognizes the Viterbi algorithm as a **maximum likelihood decoding**
1970s	Used in satellite/deep-space communications Viterbi decoding implemented for small memory orders
1990s	Applied in cellular mobile communication systems

[1]

[1] David Forney Jr, "The Viterbi Algorithm: A Personal History", 2005, https://arxiv.org/pdf/cs/0504020

Convolutional Codes in Cellular Standards[2]



[2]H. Jafarkhani, H. Maleki, and M. Vaezi, "Modulation and Coding for NOMA and RSMA," Proceedings of the IEEE, vol. 112, No. 9, pp. 1179–1213, September 2024.

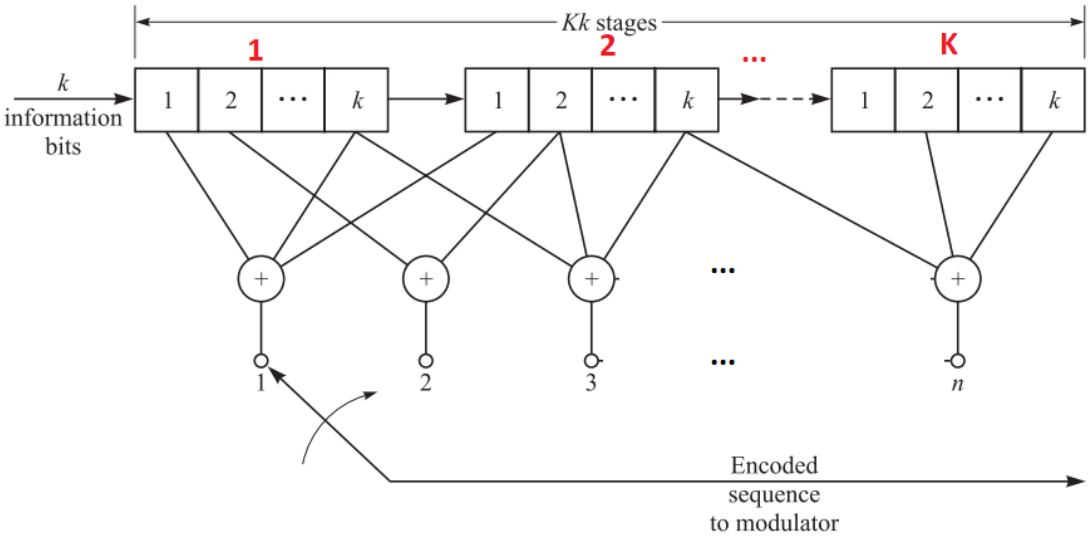
Key Points on Convolutional Codes

Convolutional codes are widely used error-correcting codes.

Unlike block codes which process fixed-size blocks, convolutional codes generate output based on current and past input bits using shift registers and generator polynomials.
Their name comes from the fact that, the encoder adds redundancy continuously through a **convolution process**.

- Convolutional codes operate on a continuous data stream, not on predefined data blocks.
- They use memory to incorporate previous bits into the encoding/decoding process.
- Encoding is done by passing the input sequence through a linear finite-state shift register.
- The encoder adds redundancy continuously through a convolution process.

2.6.2 Structure of a Convolutional Encoder

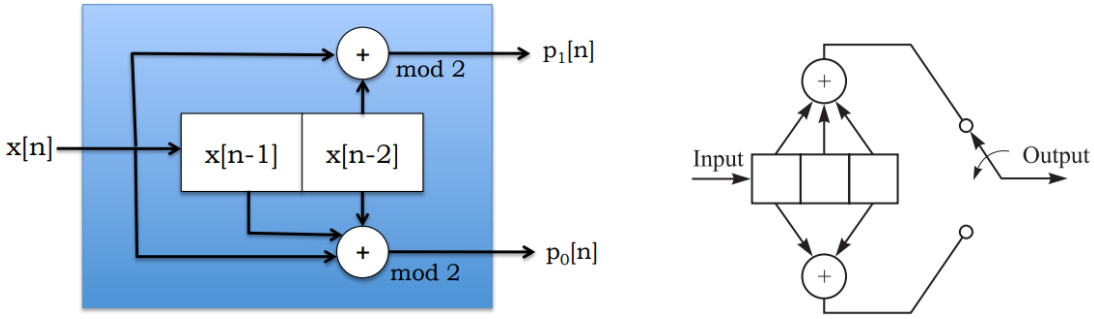


Convolutional encoder.

- A convolutional encoder is often defined by parameters n, k, and K, where
  - k: number of input bits per time unit,
  - n: number of output bits per time unit (linear algebraic function generators),
  - K: (constraint length) is the number of stages in encoding.

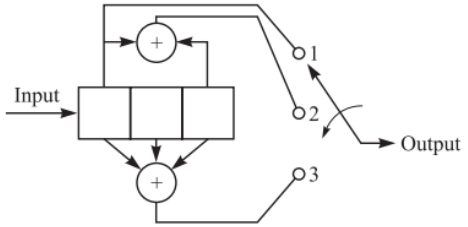
In general, the shift register consists of K stages (each with k bits–also called k-tuples) and n linear algebraic function generators.

- When $k = 1$, the data stream is processed continuously and not blocked — this is common in practice.
- Implementation relies on sequential logic circuits (due to memory requirement).
- The constraint length K is defined as the total number of shifts over which a single information bit can influence the encoder output.
  - This includes the current bit and $K - 1$ memory stages for the previous bits stored.

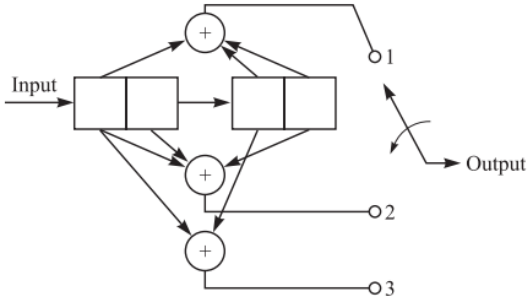


Shift-register view and its equivalent. $K = 3$ in both pictures.

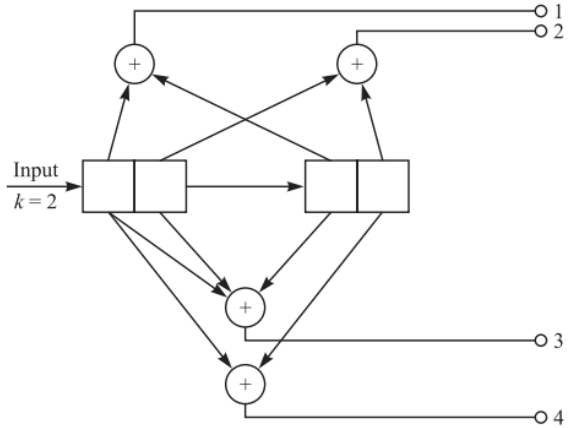
Examples



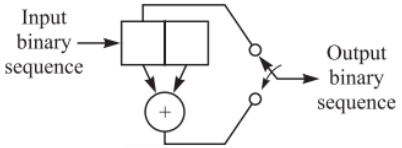
$K = 3, k = 1, n = 3$ convolutional encoder.



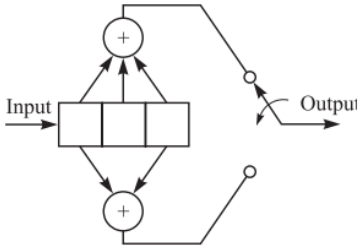
$K = 2, k = 2, n = 3$ convolutional encoder.



$K = 2, k = 2, n = 4$ convolutional encoder.



rate 1/2, constraint length $K = 2$, convolutional code.



$k = 1$, $K = 3$, and $n = 2$ convolutional code

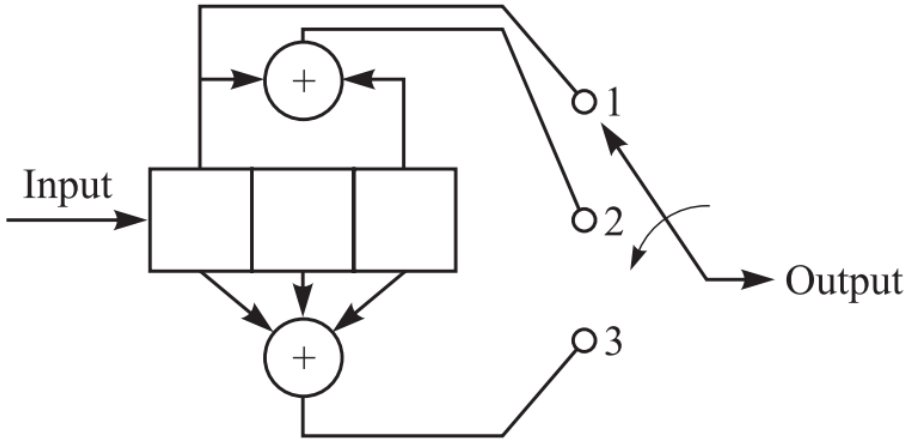
Convolutional encoders with different values of k, n, and K.

Basic Convolutional Encoder

A typical convolutional encoder consists of

- A shift register with memory elements.
- A set of modulo-2 adders (XOR gates).
- n generator polynomials defining connections from input and memory bits to the outputs.

- **Example:** The rate $R = 1/3$ encoder below



$K = 3, k = 1, n = 3$ convolutional encoder.

– could be described by the generator polynomials

$$g_1(D) = 1, \quad g_2(D) = 1 + D^2, \quad g_3(D) = 1 + D + D^2,$$

or binary vector form

$$\mathbf{g}_1 = [1\ 0\ 0], \quad \mathbf{g}_2 = [1\ 0\ 1], \quad \mathbf{g}_3 = [1\ 1\ 1]$$

– This means **three output bits** are generated for every one input bit.

– $k = 1$ and code rate is $R = n/k = 1/3$

- $\mathbf{g}_1$, $\mathbf{g}_2$, and $\mathbf{g}_3$ are the impulse responses of the encoder input to the three outputs.
- If the input sequence is $\mathbf{m}$, then the outputs are

$$\mathbf{c}^{(1)} = \mathbf{m} \star \mathbf{g}_1$$
$$\mathbf{c}^{(2)} = \mathbf{m} \star \mathbf{g}_2$$
$$\mathbf{c}^{(3)} = \mathbf{m} \star \mathbf{g}_3$$

  where $\star$ denotes the convolution operation.

- The overall code sequence $\mathbf{c}$ is formed by interleaving the outputs:

$$\mathbf{c} = (c_1^{(1)}, c_1^{(2)}, c_1^{(3)}, c_2^{(1)}, c_2^{(2)}, c_2^{(3)}, \dots)$$

  where
  - the subscript indicates the time index (at each time there are three outputs)
  - the superscript indicates the output (function) index

Complete convolutional codeword

- The encoder processes k input bits per time step and generates n output bits.
- Given L input time steps, the total number of input (message) bits is kL.
- The encoder has **memory** and uses the current and previous $K-1$ input blocks to generate each output.
- As a result, even after the input sequence ends, the encoder continues producing output for $K-1$ steps (to flush the memory).
- Therefore, the total number of output steps becomes $L+K-1$.
- Since n output bits are produced at each step, the total number of output bits (i.e., the length of the complete convolutional codeword) is

$$\text{complete codeword length} = n(L+K-1)$$

- This length accounts for both the actual coded data and the tail bits due to encoder memory.
- Then, $\text{code rate} = \frac{\text{message length}}{\text{complete codeword length}} = \frac{kL}{n(L+K-1)} \approx \frac{k}{n}$ for large L's

Example 6. *For a* $(2, 1, 3)$ *convolutional code (i.e.,* $n = 2$*,* $k = 1$*,* $K = 3$*), calculate the output of an encoder with*

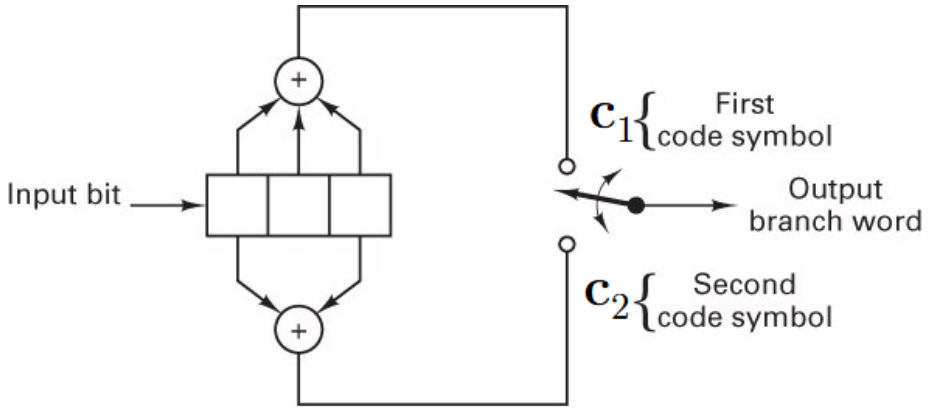
- *Generator polynomials:* $\mathbf{g}_1 = [1\ 1\ 1]$ *and* $\mathbf{g}_2 = [1\ 0\ 1]$
- *Input bits* $\mathbf{m} = [1, 0, 1, 1]$*, and shift register is initialized to 0*

Solution: Input steps are $L = 4$ (=message length since $k = 1$). Thus,

$$\text{complete codeword length} = 2(4 + 3 - 1) = 12 \text{ bits}$$

- **Step-by-step encoding**
  - Input 1: Register = [1 0 0], Output = [1, 1]
  - Input 0: Register = [0 1 0], Output = [1, 0]
  - Input 1: Register = [1 0 1], Output = [0, 0]
  - Input 1: Register = [1 1 0], Output = [0, 1]
  - Input 0 (no input): Register = [0 1 1], Output = [0, 1]
  - Input 0 (no input): Register = [0 0 1], Output = [1, 1]
- Total encoded output: $[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1]$

- **Example:** The rate $R = 1/2$ encoder below



$K = 3, k = 1, n = 2$ convolutional encoder.

– could be described by the generator polynomials

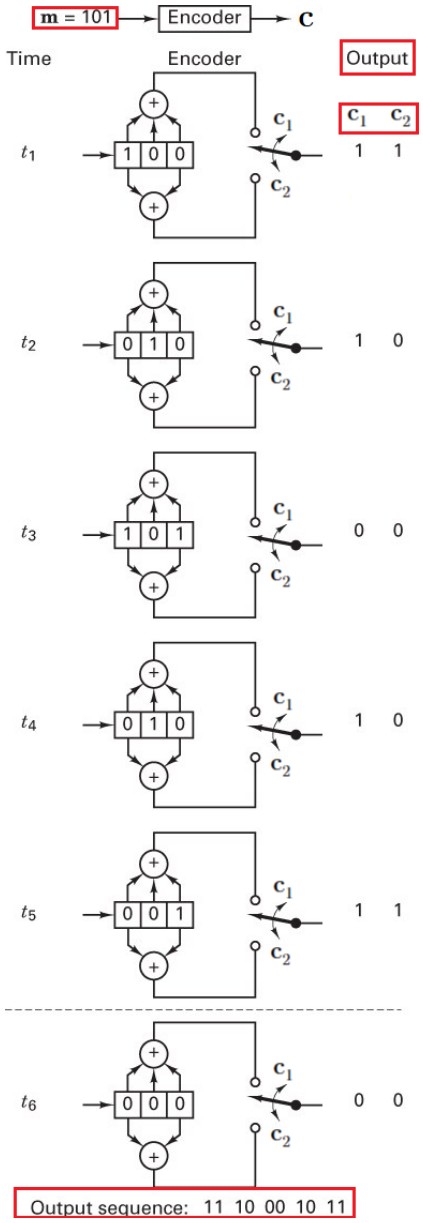
$$g_1(D) = 1 + D + D^2, \quad g_2(D) = 1 + D^2$$

or binary vectors

$$\mathbf{g}_1 = [1\ 1\ 1], \quad \mathbf{g}_2 = [1\ 0\ 1]$$

– This means **two output bits** are generated for every one input bit.

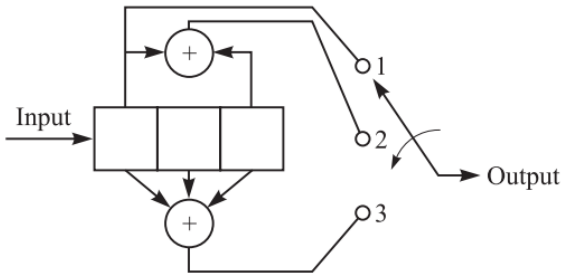
– $k = 1$ and code rate is $R = n/k = 1/2$


m = 101
Encoder
c
Time
Encoder
Output
c_1 c_2
t_1
1 0 0
1 1
t_2
0 1 0
1 0
t_3
1 0 1
0 0
t_4
0 1 0
1 0
t_5
0 0 1
1 1
t_6
0 0 0
0 0
Output sequence: 11 10 00 10 11

Several methods are used to describe a convolutional encoder:

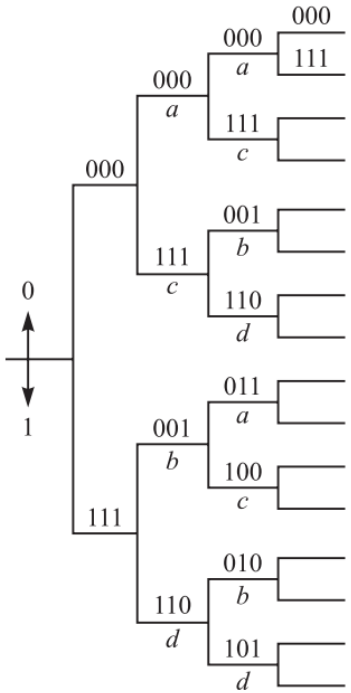
- Connection pictorial (shift-register)
- Connection vectors or polynomials
- State diagram
- Tree diagram
- Trellis diagram

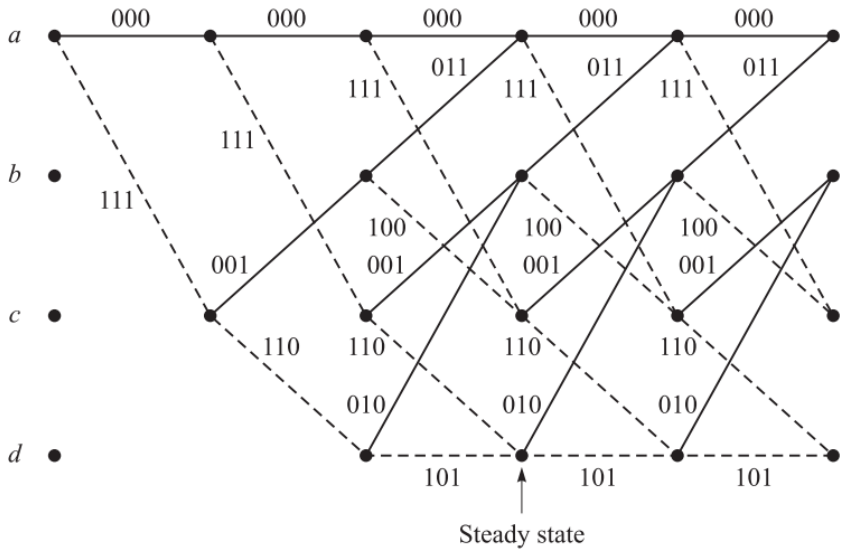
- We are going to explore the last three representations in the following
  - The state diagram shows all possible encoder states (memory contents) and transitions between them.
  - The trellis diagram unfolds these transitions across time steps, showing valid output sequences.

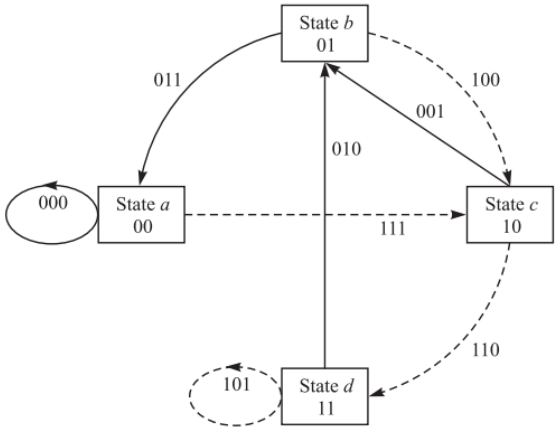


Example (Proakis)

$K = 3, k = 1, n = 3$ convolutional encoder.







Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i	
				$\mathbf{c}_1$	$\mathbf{c}_2$
—	0 0 0	0 0	0 0	—	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

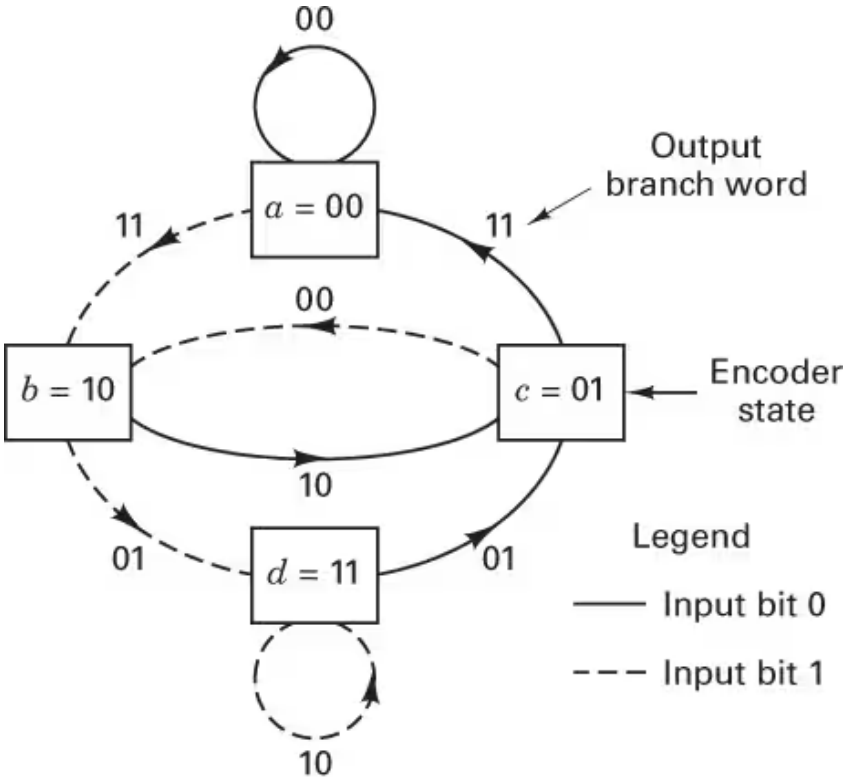
state t_{i+1}

- The state of the encoder is defined as the content of the rightmost $K-1$ k-tuples.
- Recall that the first k-tuple is the input.
- In this example, $k = 1$ and $K = 3$, thus the encoder state is the two the rightmost bits, and the input is the first bit.

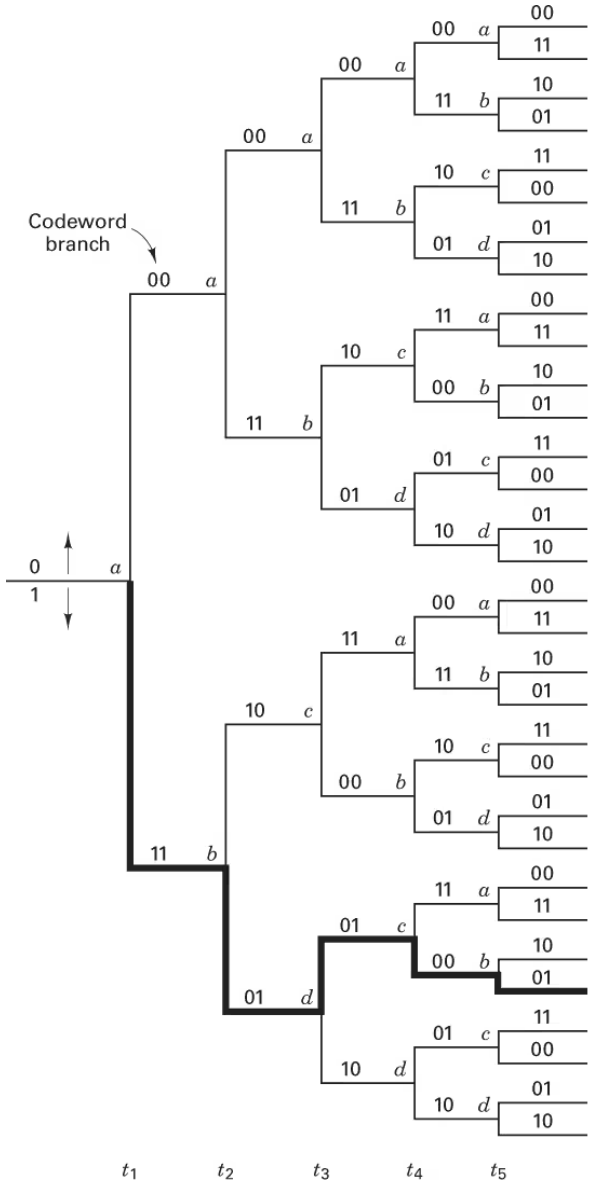
State Diagram

- The state of the encoder is defined as its shift register contents.
- In general, there are $2^{k(K-1)}$ states.
- Also, there are 2^k branches entering/leaving each states

  Example: For a 2-memory encoder ($k = 1, K = 3$), there are $2^2 = 4$ states.

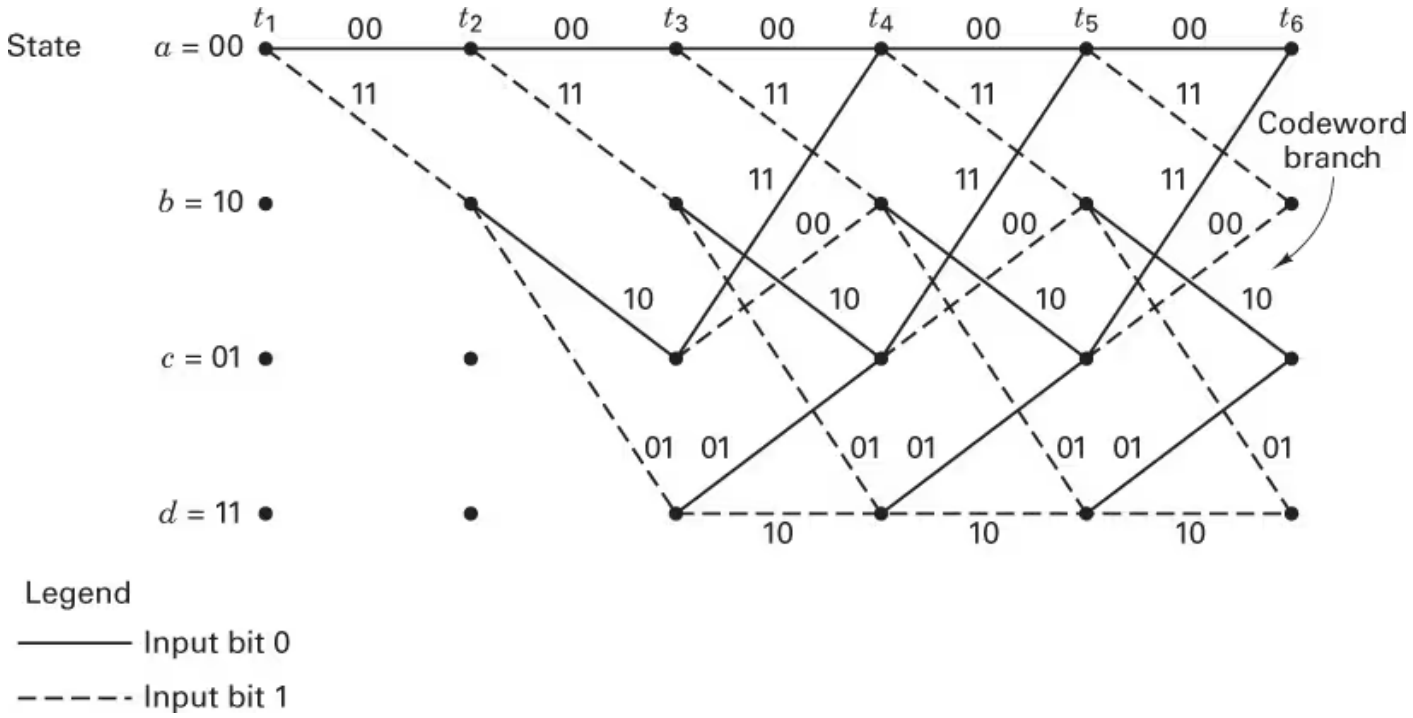


Tree diagram



Trellis diagram

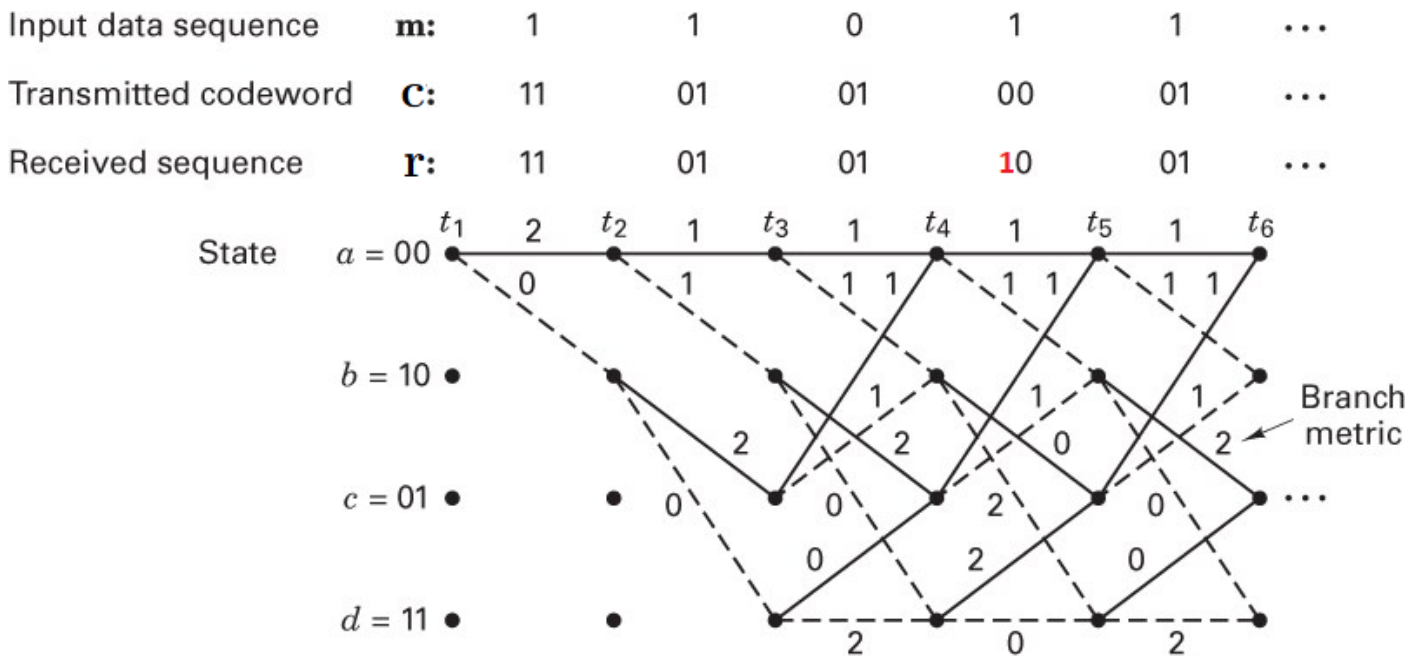
- The convolution encoder can be completely characterized by the state diagram except that it cannot be used easily to track the encoder transitions as a function of time.
- Trellis diagram is then devised for this purpose.

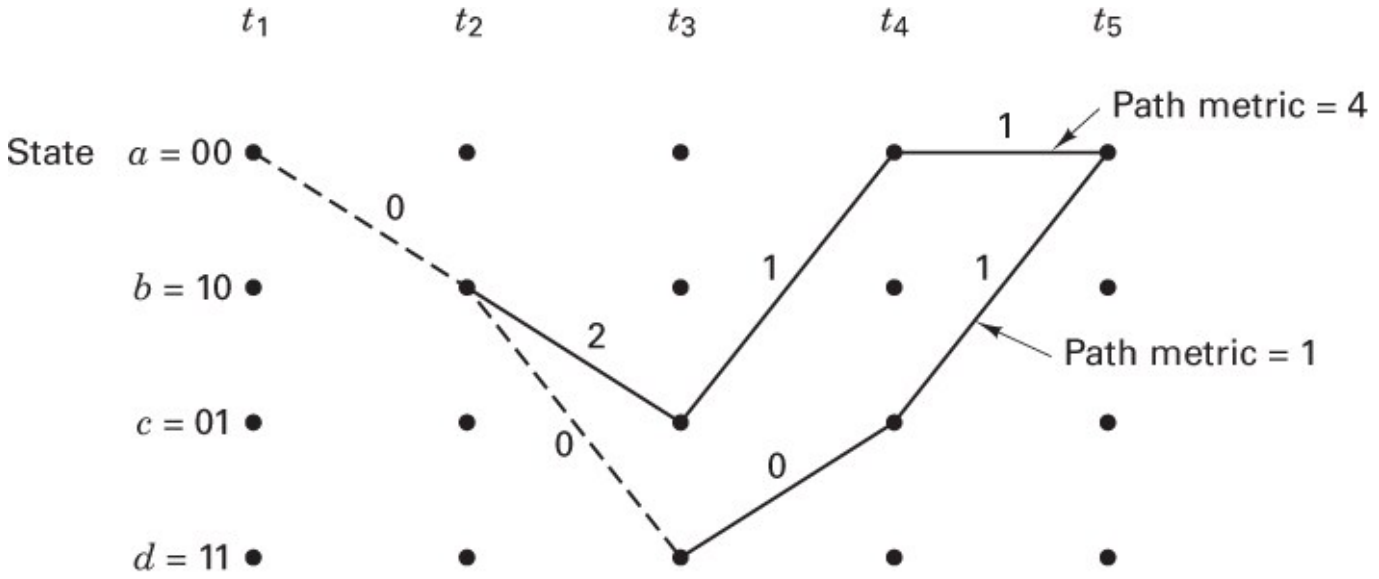


2.6.3 Viterbi Decoding Algorithm

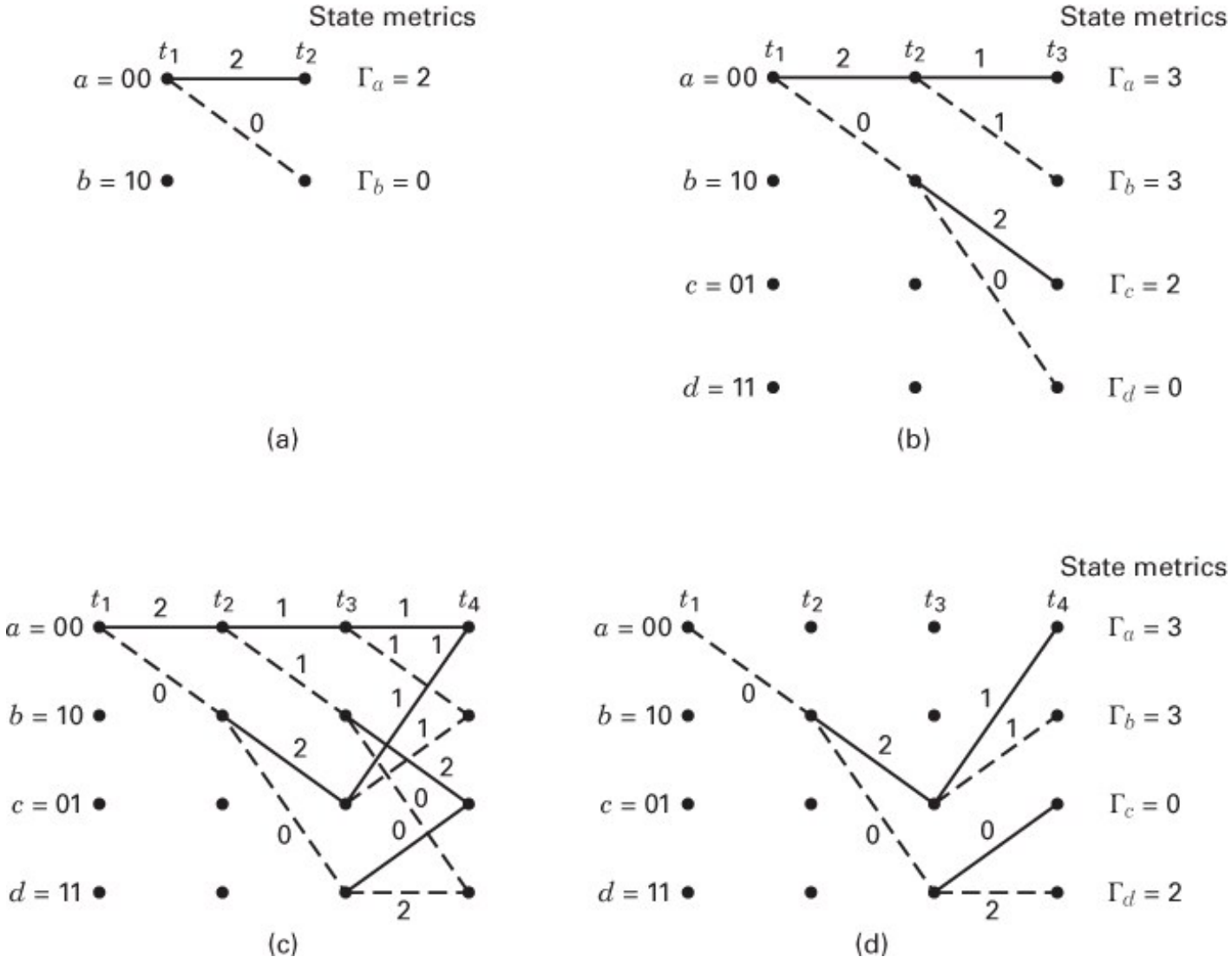
Viterbi decoding is a maximum likelihood algorithm that traces the most probable path through a trellis diagram.

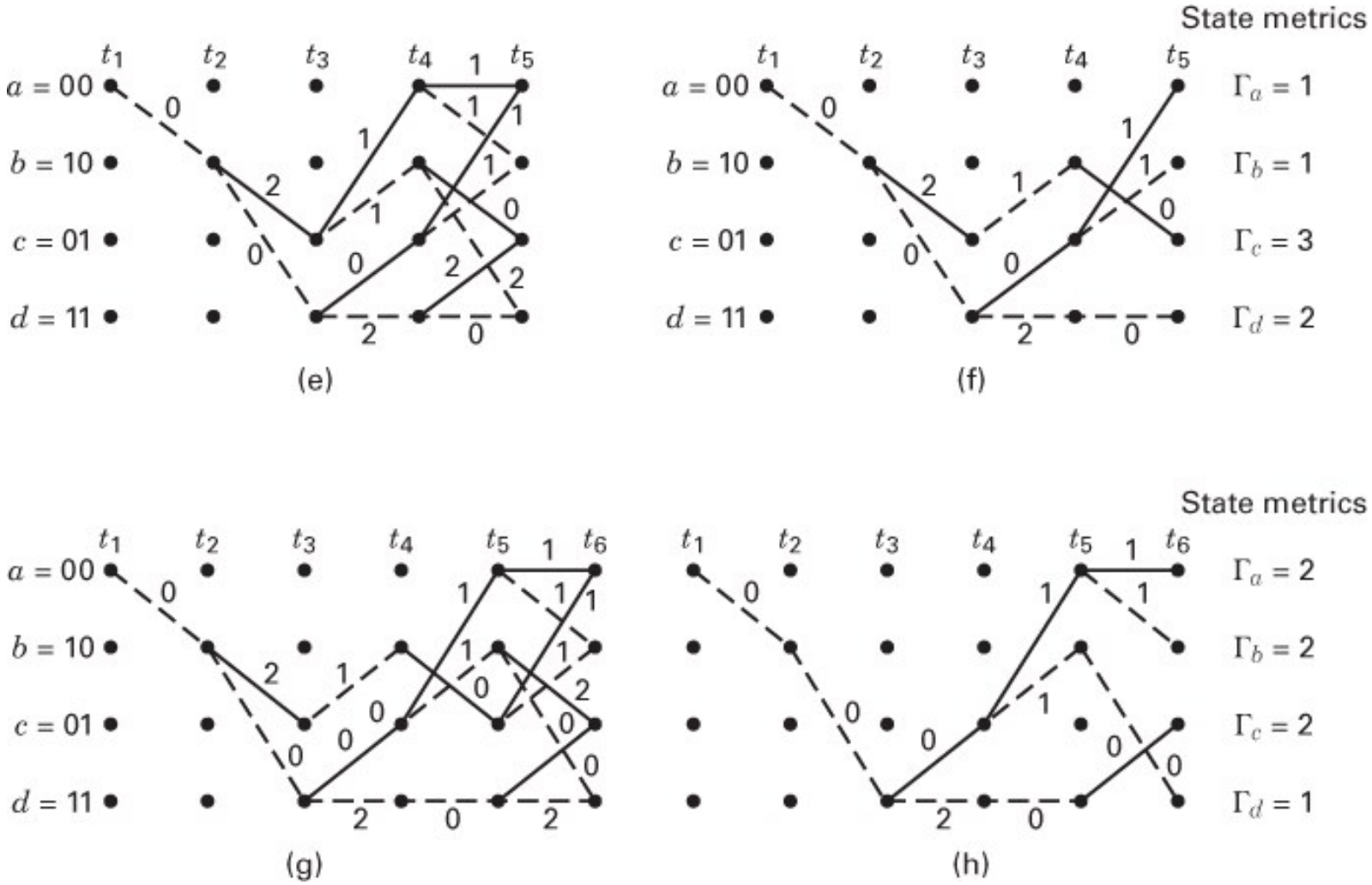
- Each path in the trellis has an associated metric (distance from received sequence).
- At each step, paths are extended and the best path to each state is retained.
- At the end, the path with the minimum cumulative metric is selected.





- When two paths enter the same state, the one with the smaller metric is selected– this is the **surviving path**.
- Surviving paths are selected for all states in the trellis.
- The decoder proceeds through the trellis, eliminating less likely paths at each step.
- Early rejection of unlikely paths reduces decoding complexity.

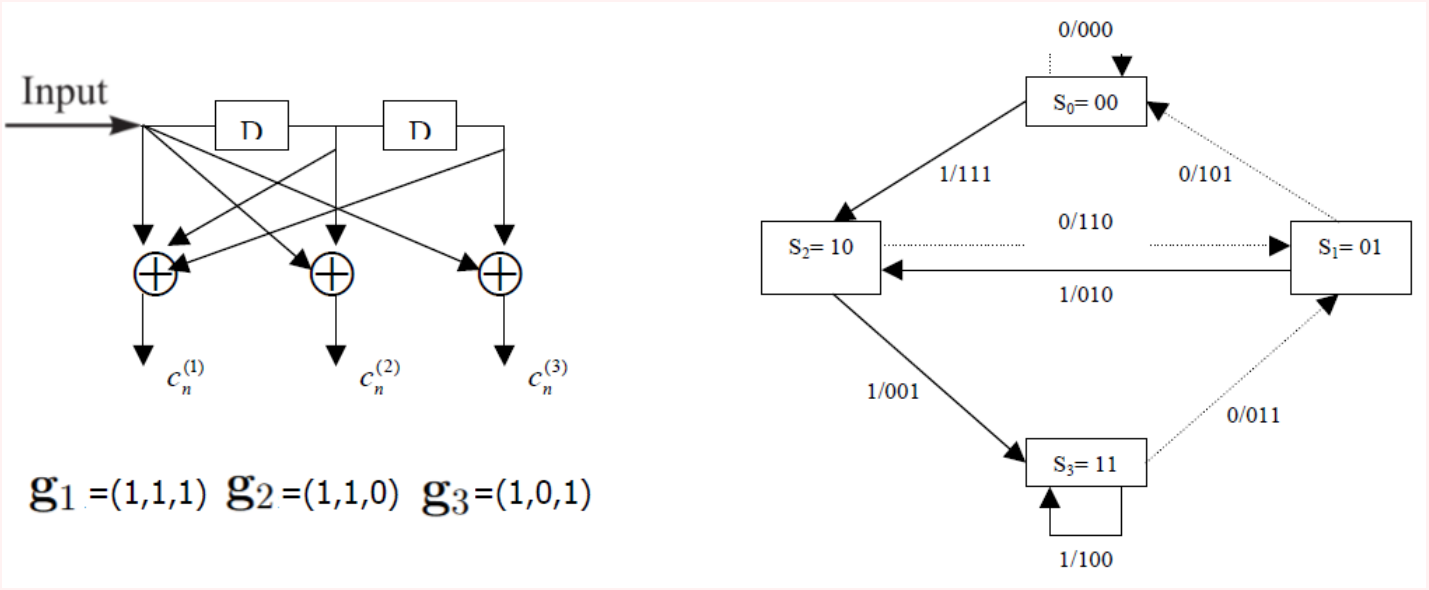

State metrics
t_1
t_2
t_3
t_4
$a = 00$
$b = 10$
$c = 01$
$d = 11$
$\Gamma_a = 2$
$\Gamma_b = 0$
$\Gamma_a = 3$
$\Gamma_b = 3$
$\Gamma_c = 2$
$\Gamma_d = 0$
$\Gamma_a = 3$
$\Gamma_b = 3$
$\Gamma_c = 0$
$\Gamma_d = 2$
(a)
(b)
(c)
(d)


State metrics
t_1 t_2 t_3 t_4 t_5 t_6
$a = 00$
$b = 10$
$c = 01$
$d = 11$
$\Gamma_a = 1$
$\Gamma_b = 1$
$\Gamma_c = 3$
$\Gamma_d = 2$
(e)
(f)
$\Gamma_a = 2$
$\Gamma_b = 2$
$\Gamma_c = 2$
$\Gamma_d = 1$
(g)
(h)

Viterbi Examples

Example 7. *Rate 1/3 convolutional code.*

- *Show that for the input information bit strea* $\mathbf{m} = 100011$, *the transmitted coded sequence is* $\mathbf{c} = 111\ 110\ 101\ 000\ 111\ 001$
- *Apply Viterbi decoding algorithm to estimate the tranmitted sequence if the received sequence is* $\mathbf{r} = 101\ 100\ 101\ 001\ 111\ 001$



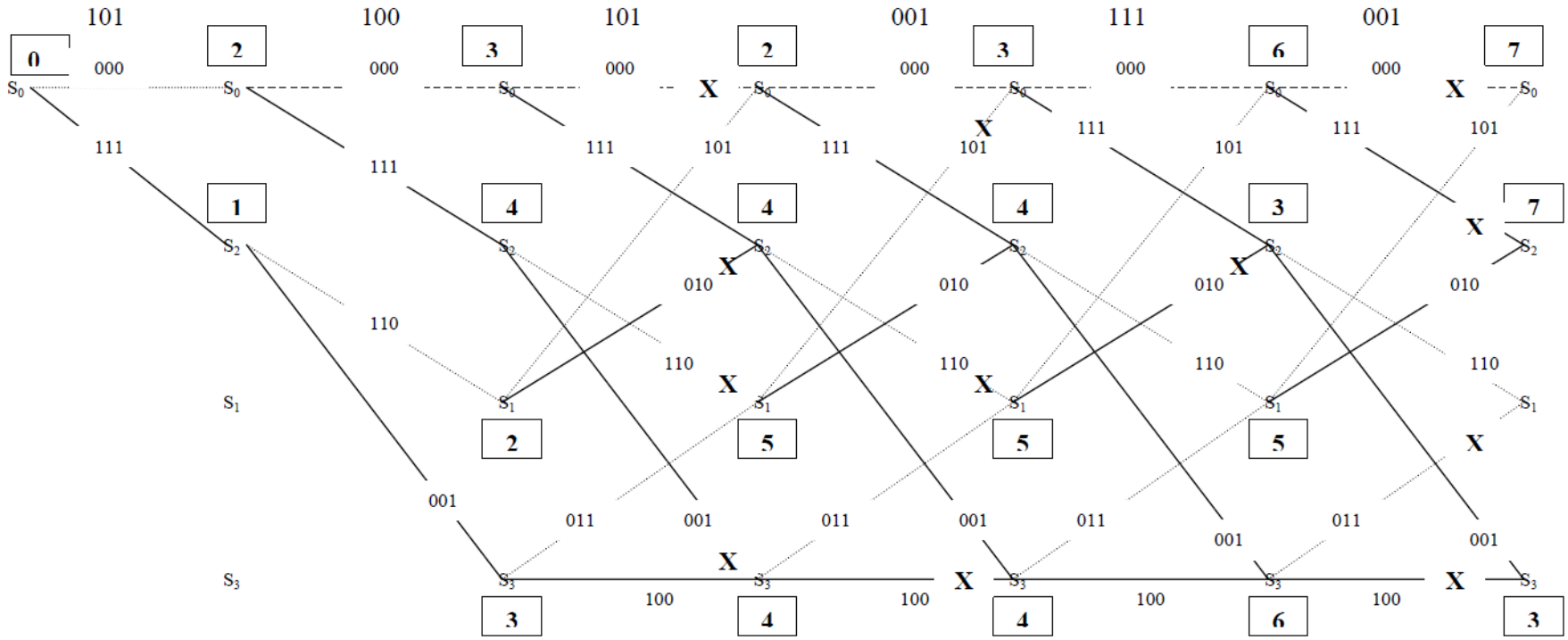
Solution:

- Branch metric (or weight): Use the Hamming distance between the received symbol and the branch output pattern.
- Move **from left to right** across the trellis.
- At each node:

  (i) Compute the *accumulated metric* for each incoming branch as the sum of
  - The metric of the previous node, and
  - The branch metric of the current transition.

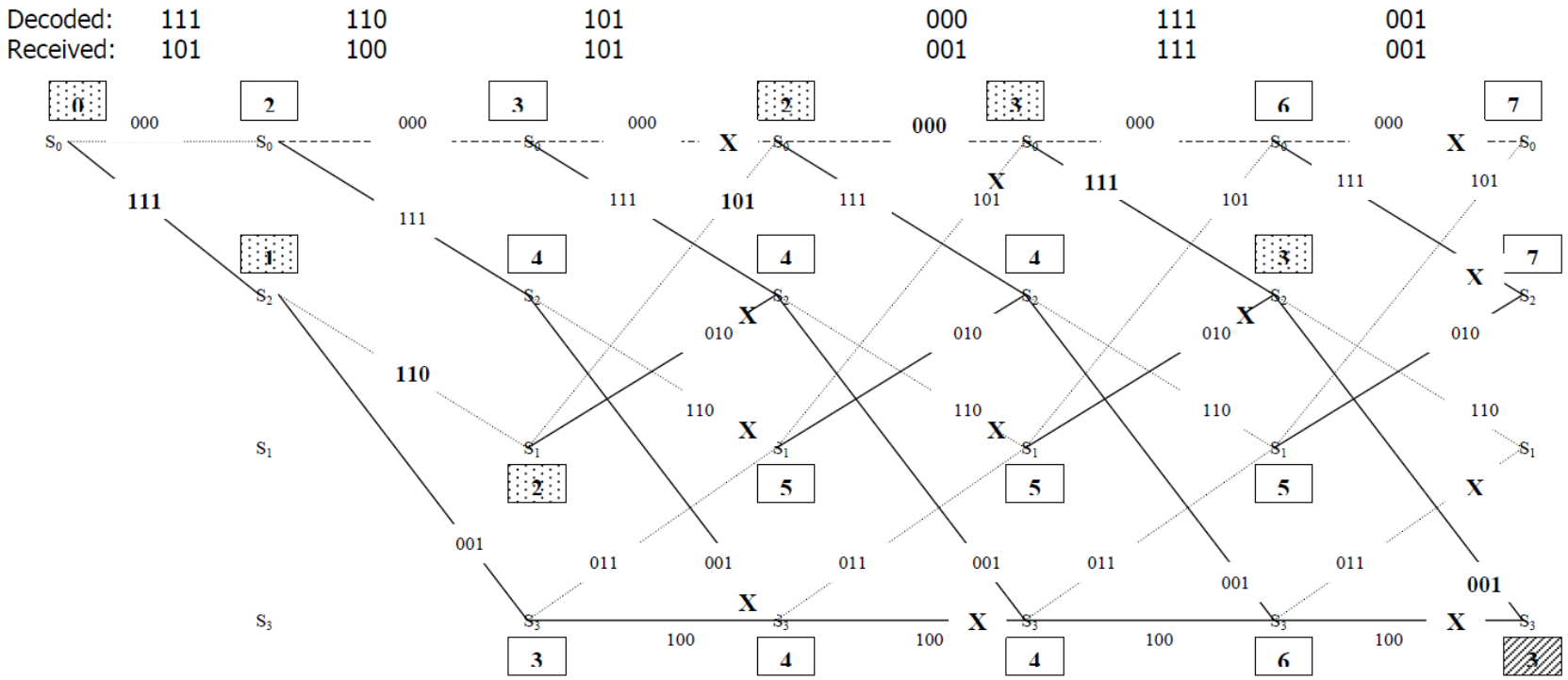
  (ii) *Select the smallest* accumulated metric
  - Assign it as the current node's metric.
  - Mark the corresponding branch as the **survivor**.
  - Discard other branches (cross them out).

Step 1: Accumulated metrics evaluation and pruning

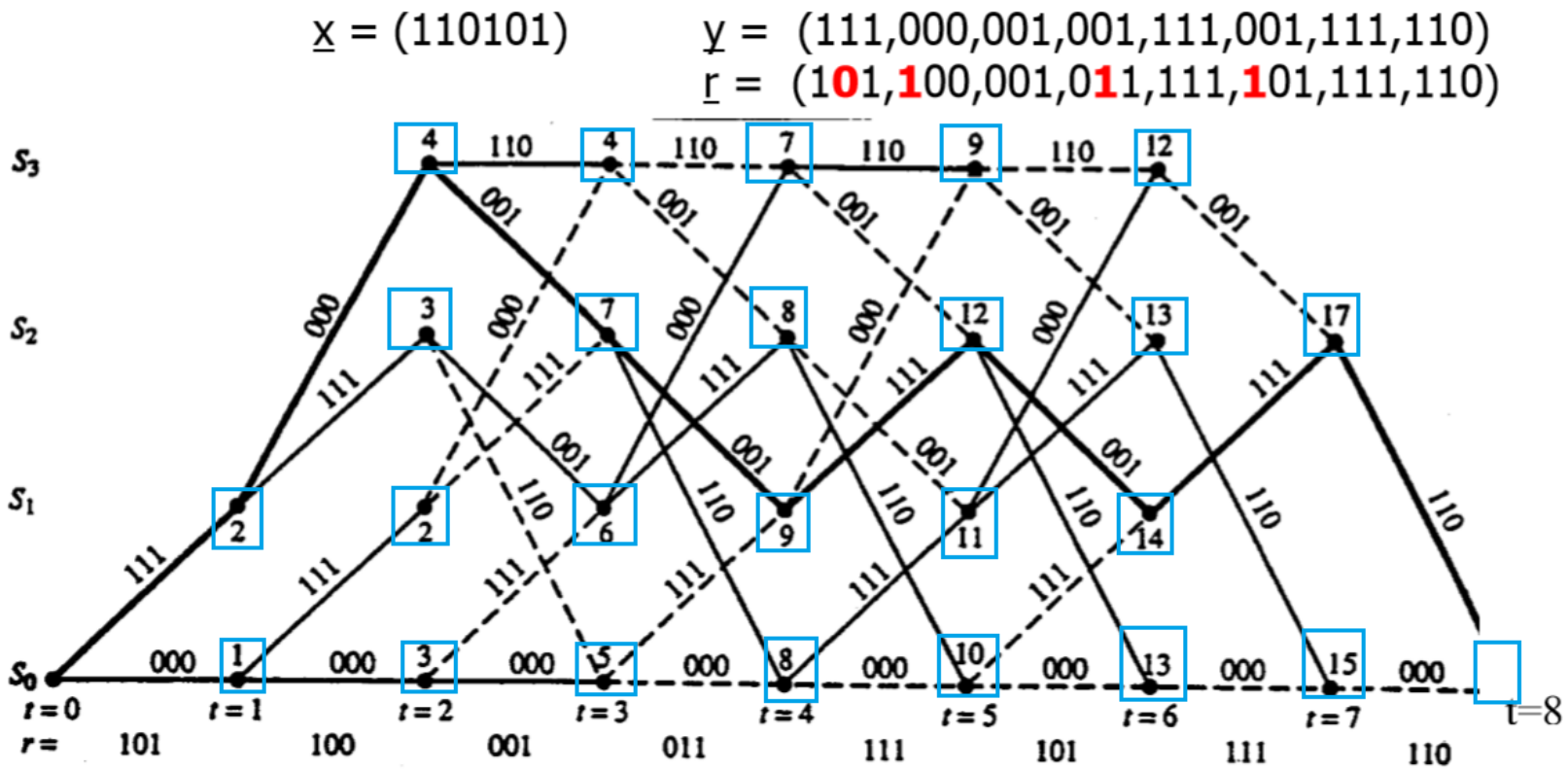


Step 2: Backtracking

- Start from the **rightmost time step** in the trellis.
- Identify the node with the **lowest metric**.
- **Back-trace** through the trellis by following the **survivor branches**.
- The traced path corresponds to the ML decoded sequence.



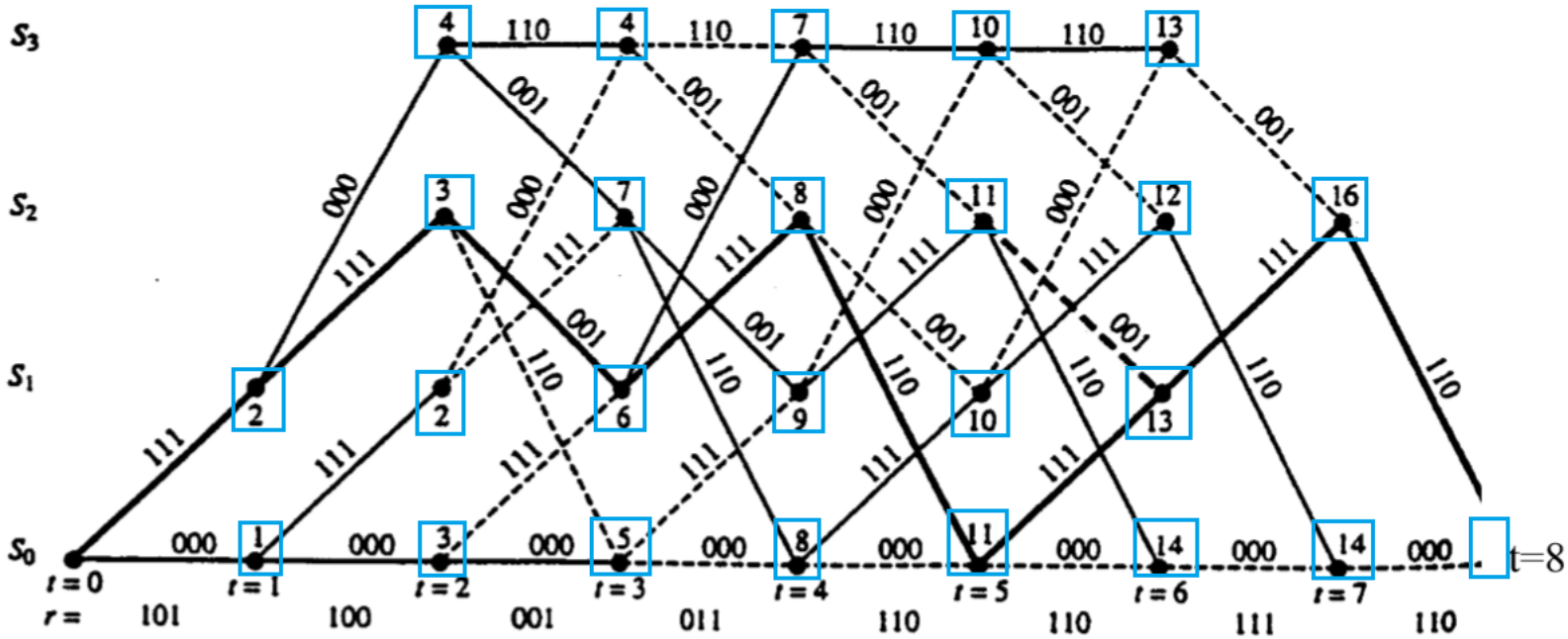
Example 2



- **x** is the message (**m**),
- **y** is the codeword (**c**), and
- **r** is the received (demodulated) sequence.

Example 3

y = (111,000,001,001,111,001,111,110)
r = (101,100,001,011,110,110,111,110) burst of 4 errors



y′ = (111,111,001,111,110,111,111,110) x′ = (1 0 1 0 0 1): wrong

- Bursty patterns typically cannot be corrected.

2.6.4 Convolutional Code Decoding Details

- **Message Sequence**
  - consists of L symbols, with each symbol composed of k bits.

$$\mathbf{m} = \underbrace{m_1^{(1)} m_1^{(2)} \cdots m_1^{(k)}}_{\mathbf{m}_1} \cdot \underbrace{m_2^{(1)} m_2^{(2)} \cdots m_2^{(k)}}_{\mathbf{m}_2} \cdots \underbrace{m_i^{(1)} m_i^{(2)} \cdots m_i^{(k)}}_{\mathbf{m}_i} \cdots \underbrace{m_L^{(1)} m_L^{(2)} \cdots m_L^{(k)}}_{\mathbf{m}_L}$$

  - i^{th} message symbol is $\mathbf{m}_i = m_i^{(1)} m_i^{(2)} \cdots m_i^{(k)}$, where $m_i^{(j)} \in \{0, 1\}$ is a single bit.
  - for each encoder clock cycle, one message symbol (of k bits) enters the encoder.
  - there are 2^{kL} possible message sequences (and thus that many code sequences).

- **Code Sequence**
  - consists of N_c symbols, with each symbol composed of n bits.

$$\mathbf{c} = \underbrace{c_1^{(1)} c_1^{(2)} \cdots c_1^{(n)}}_{\mathbf{c}_1} \cdot \underbrace{c_2^{(1)} c_2^{(2)} \cdots c_2^{(n)}}_{\mathbf{c}_2} \cdots \underbrace{c_i^{(1)} c_i^{(2)} \cdots c_i^{(n)}}_{\mathbf{c}_i} \cdots \underbrace{c_{N_c}^{(1)} c_{N_c}^{(2)} \cdots c_{N_c}^{(n)}}_{\mathbf{c}_{N_c}}$$

  - i^{th} code symbol is $\mathbf{c}_i = c_i^{(1)} c_i^{(2)} \cdots c_i^{(n)}$, where $c_i^{(j)} \in \{0, 1\}$ is a single bit.

- for each encoder clock cycle, one code symbol (of n bits) is produced by the encoder.
- $N_c = L + (K-1)$ where K is the constraint length of the code.
- The extra $(K-1)$ symbols arise as the encoder is forced back to the all-zero state.

- **Received Sequence**
  - also consists of N_c symbols, with each symbol composed of n bits.

$$\mathbf{r} = \underbrace{r_1^{(1)} r_1^{(2)} \cdots r_1^{(n)}}_{\mathbf{r}_1} \cdot \underbrace{r_2^{(1)} r_2^{(2)} \cdots r_2^{(n)}}_{\mathbf{r}_2} \cdots \underbrace{r_i^{(1)} r_i^{(2)} \cdots r_i^{(n)}}_{\mathbf{r}_i} \cdots \underbrace{r_{N_c}^{(1)} r_{N_c}^{(2)} \cdots r_{N_c}^{(n)}}_{\mathbf{r}_{N_c}}$$

  - i^{th} received symbol is $\mathbf{r}_i = r_i^{(1)} r_i^{(2)} \cdots r_i^{(n)}$, where $r_i^{(j)} \in \{0, 1\}$ is a single bit.
  - j^{th} bit of the i^{th} symbol is given by $r_i^{(j)} = c_i^{(j)} \oplus e_i^{(j)}$, where $e_i^{(j)}$ is the bit error indicator which is equal to 1 if the bit is in error, and 0 otherwise.
  - the channel transition probability is

$$\Pr\left\{ r_i^{(j)} | c_i^{(j)} \right\} = \begin{cases} 1-p, & \text{if } r_i^{(j)} = c_i^{(j)} \\ p, & \text{if } r_i^{(j)} \neq c_i^{(j)} \end{cases}$$

Distances Between All Codewords

- There are 2^{kL} possible message sequences
  - Thus, there are also 2^{kL} code sequences or codewords (why?)
- Each of the 2^{kL} code sequences is represented by a unique path through the trellis.
- **Truncated codewords:** $\mathbf{y}_i^{(L)}$ denotes output sequences from L branches (with Lk info bits and Ln coded bits).
- **Linearity:** For linear convolutional codes, $\mathbf{y}_k^{(L)} = \mathbf{y}_i^{(L)} + \mathbf{y}_j^{(L)}$ is a valid codeword.
- **Column distance function:**
  - Defined as the minimum Hamming distance between all pairs $\mathbf{y}_i^{(L)}$, $\mathbf{y}_j^{(L)}$ where $\mathbf{x}_i^{(1)} \neq \mathbf{x}_j^{(1)}$.
  - Equivalent to: $d^{(L)} = \min \left\{ w_H \left(\mathbf{y}^{(L)}(\mathbf{x}) \right) \mid \mathbf{x}^{(1)} \neq 0 \right\}$.

- **Minimum free distance**, d_{free},
  - d_{free} is the minimum Hamming distance between all pairs of complete convolutional codewords:

$$d_{\text{free}} = \min\left\{d_H\left(\mathbf{y}_i, \mathbf{y}_j\right) \mid \mathbf{y}_i \neq \mathbf{y}_j\right\} = \min\left\{w_H\left(\mathbf{y}(\mathbf{x})\right) \mid \mathbf{x} \neq 0\right\}$$

  - For **non-catastrophic** convolutional codes

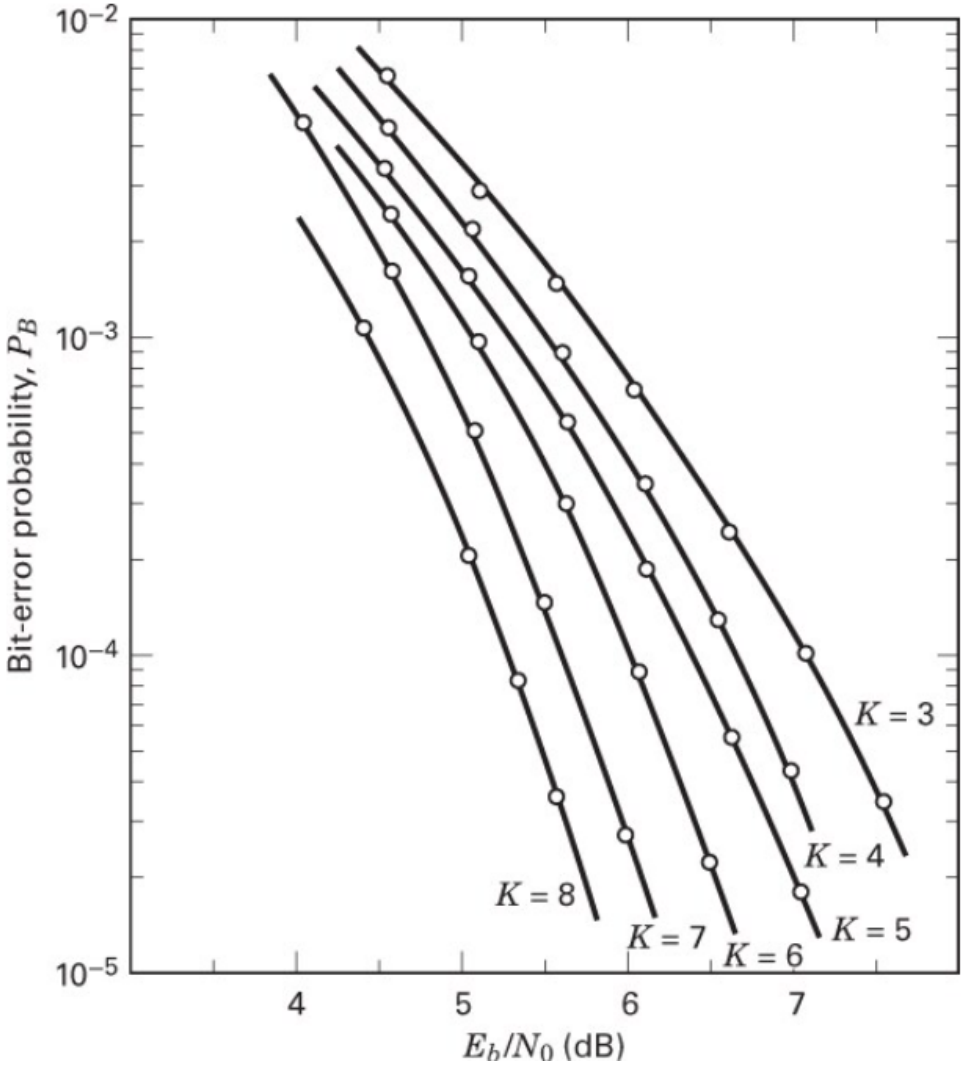
$$d^{(L)} \longrightarrow d_{\text{free}} \quad \text{as} \quad L \rightarrow \infty$$

  - Probability of error is smaller when d_{free} is large

Table 2: **Best known d_{free}**

K	RATE-1/3		RATE-1/2	
	Systematic	Non-Systematic	Systematic	Non-Systematic
2	5	5	3	3
3	6	8	4	5
4	8	10	4	6
5	9	12	5	7
6	10	13	6	8
7	12	15	6	10

The Effect of the Constraint Length K



Memory vs. Performance

- $K - 1$ is the number of memory stages
  - $K \uparrow \Longrightarrow d_{\text{free}} \uparrow$
  - $d_{\text{free}} \uparrow \Longrightarrow$ BER $\downarrow$
- But, the decoding complexity increase exponentially with K

TABLE 8.3–1
Rate 1/2 Maximum Free Distance Codes

Constraint Length K	Generators in Octal		d_{free}	Upper Bound on d_{free}
3	5	7	5	5
4	15	17	6	6
5	23	35	7	8
6	53	75	8	8
7	133	171	10	10
8	247	371	10	11
9	561	753	12	12
10	1,167	1,545	12	13
11	2,335	3,661	14	14
12	4,335	5,723	15	15
13	10,533	17,661	16	16
14	21,675	27,123	16	17

Sources: Odenwalder (1970) and Larsen (1973).

2.6.5 Soft-decision (Soft-input) Decoding*

- The previously discussed decoder is called **hard-decision** (hard-input) decoder
  - Operates on binary decisions from the receiver's decision device (BSC output)
- **Soft-decision** (soft-input) decoding offers better performance
  - Uses analog output from receiver's matched filter (AWGN channel)
  - Preserves reliability information about each bit
  - Typically provides 2-3 dB coding gain over hard-decision
- For soft decoding, we may define a channel with four outputs: $0, \underline{0}, 1, \underline{1}$
  - Then we need four conditional probabilities for each input
    - $p(0|0), \quad p(0|1)$
    - $p(\underline{0}|0), \quad p(\underline{0}|1)$
    - $p(1|0), \quad p(1|1)$
    - $p(\underline{1}|0), \quad p(\underline{1}|1)$

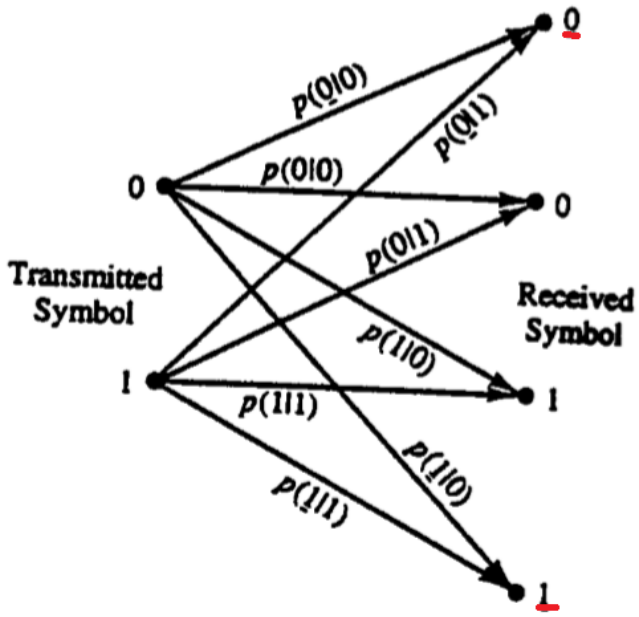


Table 3: Conditional probability $p(r|y)$

$p(r\|y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	0.50	0.32	0.13	0.05
$y = 1$	0.05	0.13	0.32	0.50

Table 4: Logarithmic conditional probability $\log_2 p(r|y)$

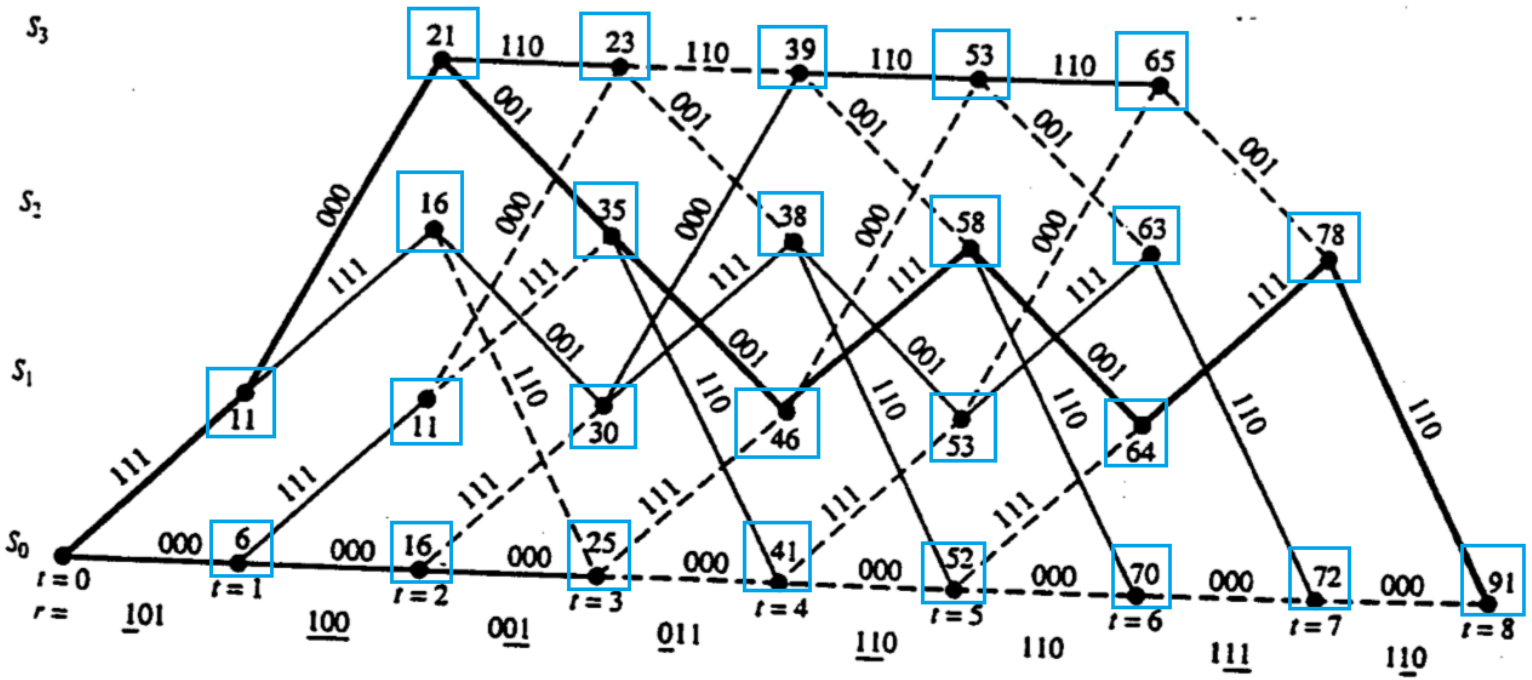
$\log_2 p(r\|y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	-1.00	-1.64	-2.94	-4.32
$y = 1$	-4.32	-2.94	-1.64	-1.00

Table 5: Metric calculation $M(r|y) = 1.5[\log_2 p(r|y) - \log_2(0.05)]$

$M(r\|y)$	$r = \underline{0}$	$r = 0$	$r = 1$	$r = \underline{1}$
$y = 0$	5	4	2	0
$y = 1$	0	2	4	5

Soft-decoding Example:

Tx: $\underline{y}$ = (111, 000,001,001,111,001,111,110)
Rx: $\underline{r}$ = (101,100,001,011,110,110,111,110)



– The inclusion of soft-decision information enables the correction of the burst error that led to decoding errors in the third hard-decision example.

2.7 Transfer Function & Weight Distribution Polynomial

2.7.1 Weight Distribution Polynomial for Linear Block Codes

- An (n, k) code has 2^k codewords with weights between 0 and n.
- In any linear block code
  - There exists exactly one codeword of weight 0.
  - The weights of nonzero codewords can range between $d_{\min}$ and n.
- The `weight distribution polynomial` or `weight enumeration function` specifies the number of codewords of each weight.
- Denoting by $A(Z)$, it is defined as

$$A(Z) = \sum_{i=0}^{n} A_i Z^i = 1 + \sum_{i=d_{\min}}^{n} A_i Z^i$$

  where:
  - A_i denotes the number of codewords of weight i.

Example 8. *Determine $A(Z)$ for the linear codes below.*

- $\{000,\ 111\}$ $\qquad A(Z) = 1 + Z^3$
- $\{0000,\ 1100,\ 0011,\ 1111\}$ $\qquad A(Z) = 1 + 2Z^2 + Z^4$

2.7.2 Transfer Function of Convolutional Codes

- The distance properties and error rate performance of a convolutional code can be obtained from its state diagram.
- Since a convolutional code is linear
  - The set of Hamming distances from the all-zero code sequence is the same as from any code sequence.
  - We assume, without loss of generality, that the all-zero code sequence is transmitted.
- Therefore, instead of studying distance properties directly, we study the **weight distribution**, as for block codes.

- The state diagram (Figure 8.1–7) is used to demonstrate the method for obtaining distance properties.
- We focus on error events where
  - The path departs from the all-zero path and returns to it for the first time.
- Branches in the state diagram are labeled as $Z^0 = 1$, Z^1, Z^2, or Z^3, where the exponent denotes the Hamming distance to the all-zero sequence.
- The self-loop at node a is eliminated because
  - It contributes nothing to the distance properties.
  - It does not represent a departure from the all-zero path.
- Node a is split into two nodes
  - One represents the input.
  - The other represents the output, corresponding to departure and return to the all-zero path.
- The resulting diagram (Figure 8.1–12) now consists of five nodes.
- This modified diagram is used to write the four state equations.

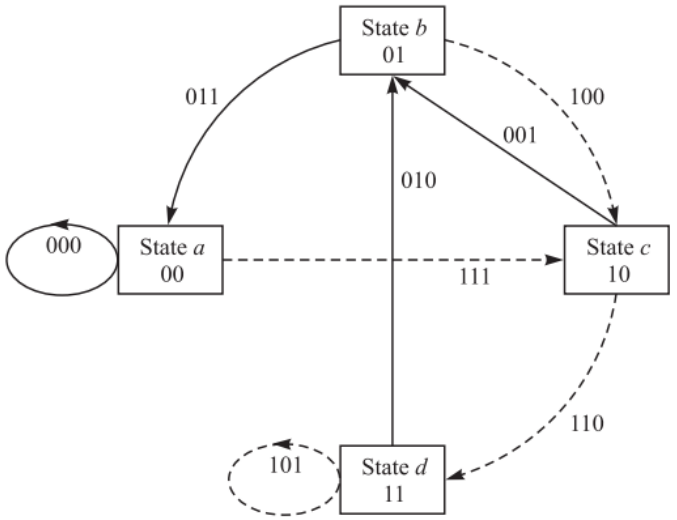


FIGURE 8.1–7
State diagram for rate $1/3$, $K = 3$ convolutional code.

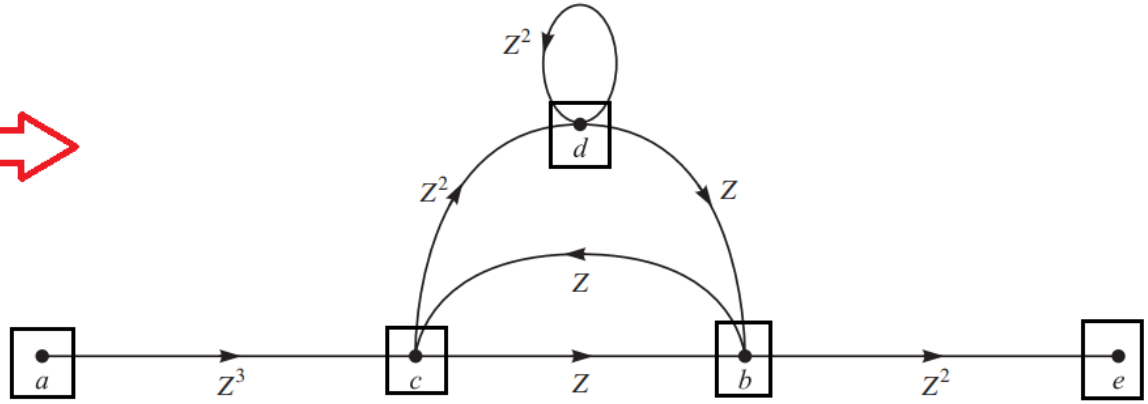


FIGURE 8.1–12
State diagram for rate $1/3$, $K = 3$ convolutional code.

– Based on the right-hand figure we have the following state equations

$$X_c = Z^3 X_a + Z X_b$$
$$X_b = Z X_c + Z X_d$$
$$X_d = Z^2 X_c + Z^2 X_d$$
$$X_e = Z^2 X_b$$

– The transfer function for the code is defined as $T(Z) = \frac{X_e}{X_a}$. Then,

$$T(Z) = \frac{X_e}{X_a} = \frac{Z^6}{1 - 2Z^2}$$
$$= Z^6 + 2Z^8 + 4Z^{10} + 8Z^{12} + \cdots$$

– The last equality uses the identity $\frac{1}{1-\alpha} = \sum_{i=0}^{\infty} \alpha^i$

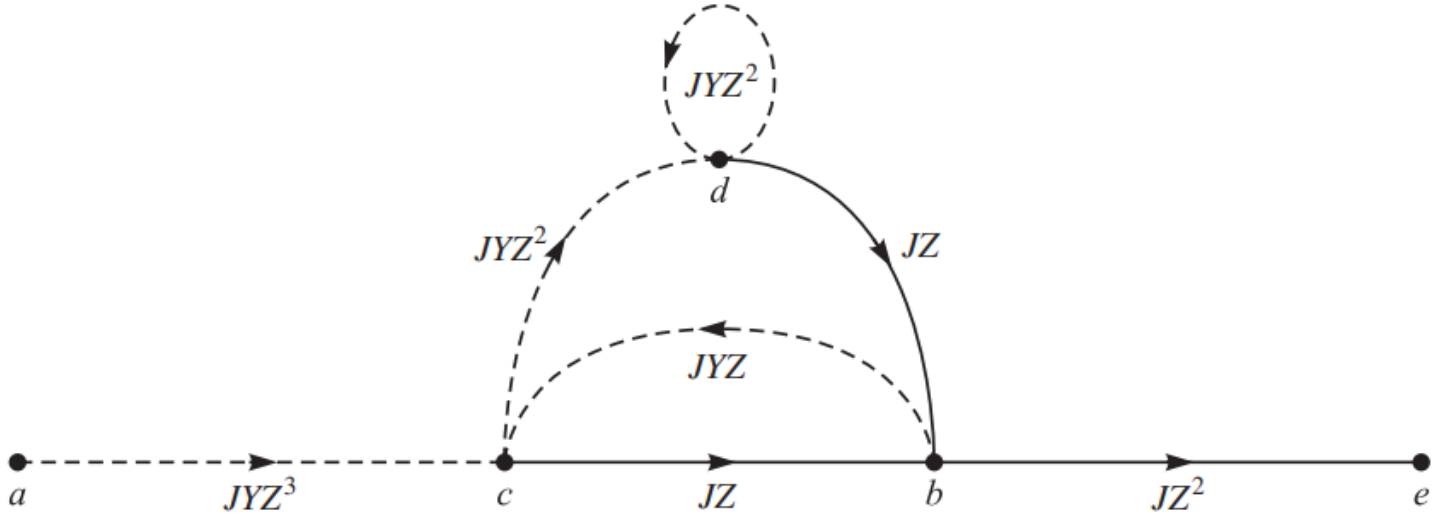
– From this transfer function, we have

* There is a single path of Hamming distance $d = 6$ from the all-zero path that returns to it. (this path is `acbe`)
* There are two paths of distance $d = 8$: `acdbe` and `acbcbe`.
* $\cdots$

Thus, the transfer function captures the distance properties of the convolutional code.

* The minimum distance of the code is called the minimum free distance, denoted by d_{free}.
* In this example, $d_{\text{free}} = 6$.

– The transfer function can provide more detailed information.



* **Exponent of Z:** branch output weight (i.e., Hamming weight of output bits)
* **Exponent of Y:** branch input weight (i.e., Hamming weight of input bits)
* **Exponent of J:** number of branches in the path (always 1 per branch)

– The state equations for the above state diagram are

$$\begin{aligned} X_c &= JYZ^3X_a + JYZX_b \\ X_b &= JZX_c + JZX_d \\ X_d &= JYZ^2X_c + JYZ^2X_d \\ X_e &= JZ^2X_b \end{aligned}$$

– Upon solving these equations for the ratio X_e/X_a, we obtain the transfer function

$$\begin{aligned} T(Y,Z,J) &= \frac{J^3YZ^6}{1-JYZ^2(1+J)} \\ &= J^3YZ^6 + J^4Y^2Z^8 + J^5Y^2Z^8 + J^5Y^3Z^{10} + 2J^6Y^3Z^{10} + J^7Y^3Z^{10} + \cdots \end{aligned}$$

– The expanded form of the transfer function $T(Y,Z,J)$ reveals detailed properties of all paths in the convolutional code.

∗ The first term in the expansion shows

· Hamming distance $d = 6$ (from exponent of Z) $\implies d_{\text{free}} = 6$,

· Path length = 3 branches (from exponent of J),

· One input bit is 1 (from exponent of Y).

  - The second and third terms $(J^4Y^2 + J^5Y^2)Z^8$ show two paths with distance $d = 8$
    - One path has length 4 (from J^4) with 2 input 1s,
    - The other has length 5 (from J^5) with 2 input 1s.
  - In general:
    - Exponent of J: number of branches (path length),
    - Exponent of Y: number of 1s in the input sequence,
    - Exponent of Z: Hamming distance of the output (codeword weight).

- The error probability is then bounded by

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d P(d)$$

  where a_d is the number of nonzero paths with weight d.

- A crude approximation is

$$P_e \approx a_{d_{\text{free}}} P(d_{\text{free}})$$

  In general, a code with a larger minimum free distance than another code will give better performance.

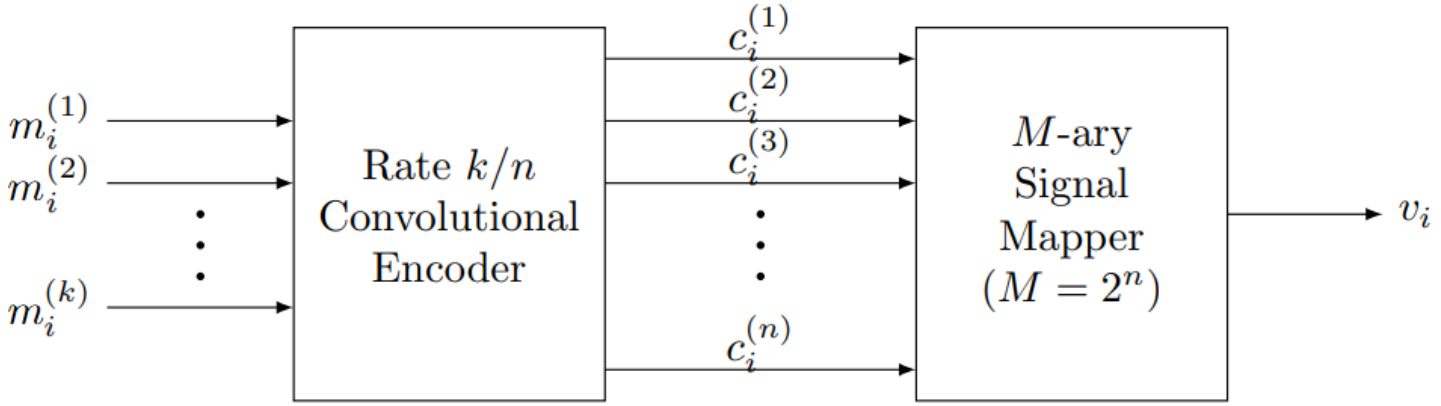
3 Trellis Coded Modulation (TCM)*

- TCM combines convolutional coding with spectrally efficient M-ary modulation.
- **Objectives:** Improve system performance without
  - increasing the transmitted energy per bit,
  - reducing the message transmission rate,
  - increasing the bandwidth.
- **Example:** Rate 1/3 convolutional code with BPSK (sending one code bit at a time)
  - Can improve performance over uncoded BPSK without increasing E_b,
  - However, to accommodate extra parity bits, one must either
    - Slow down the message transmission rate, or
    - Increase the code bit transmission rate (which increases bandwidth).

  Example: Rate 1/3 convolutional code with 8-PSK
  - Each code symbol (3 bits) is mapped to one 8-PSK signal.
  - Three code bits are sent with each channel use.

  - One message bit is transmitted per channel use.
  - No need to increase the bandwidth.
  - The message bit transmission rate remains the same as for uncoded BPSK.

- In general, TCM uses
  - A convolutional code of rate $\frac{k}{n}$,
  - An M-ary signaling scheme, where $M = 2^n$.

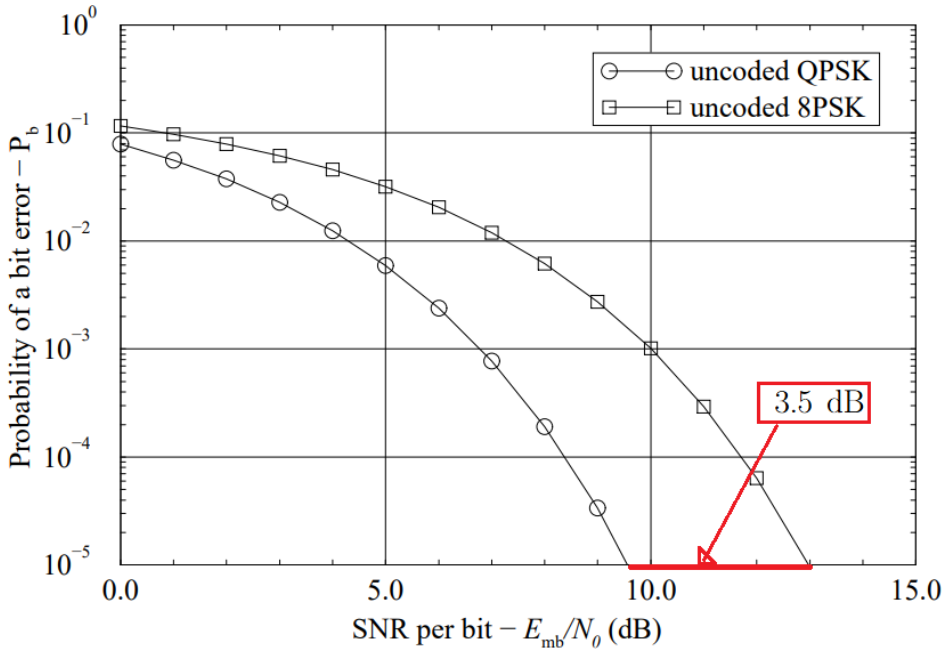


- Each n-bit code symbol is transmitted in a single use of the channel.
- TCM with a rate $\frac{k}{n}$ convolutional code must outperform uncoded modulation using $M = 2^k$ symbols.
- However, increasing M typically reduces performance (due to decreased Euclidean distance).

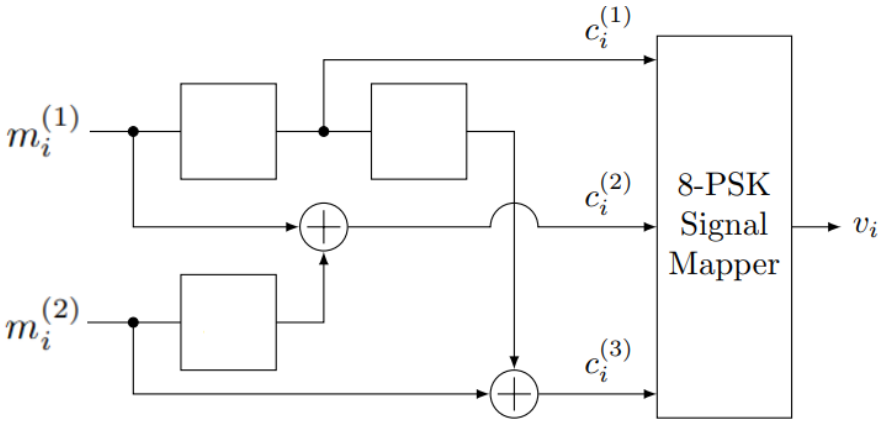
– Therefore, the convolutional code used in TCM must provide sufficient coding gain to compensate for the performance loss due to higher-order modulation.

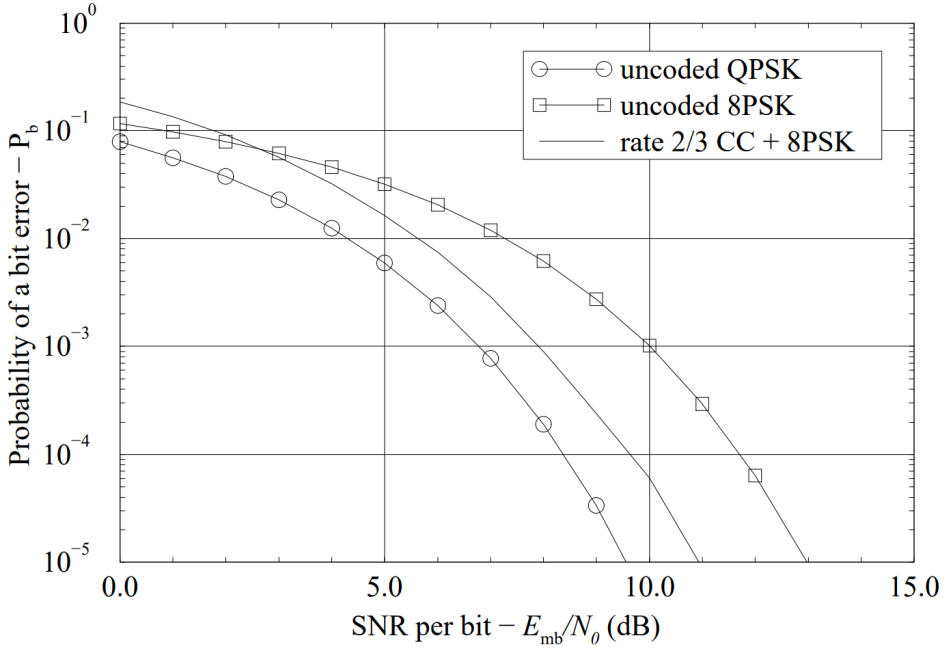
– For example, a rate $\frac{2}{3}$ convolutional code, if used with 8-PSK, must give at least 3.5 dB coding gain compared to uncoded 8-PSK to justify its use. (see the figure)

– Otherwise, uncoded QPSK is a better choice.



Example: Consider the following rate 2/3 convolutional code used with Gray-mapped 8-PSK.





- In this case, the performance of the coded scheme is worse than that of uncoded QPSK.
- There is a clear disadvantage to using coding under these conditions.
- The coding gain does not compensate for the degradation of using the higher-order modulation scheme (8-PSK).

– This degradation could be mitigated by using a more powerful code (e.g., with a longer constraint length).

– However, this would increase the decoder complexity, as it results in a higher number of states.

– The key to a well-performing coded modulation scheme is to maximize the minimum Euclidean distance **between coded signals**.

– Ungerboeck (1982) introduced a method of mapping by set partitioning to achieve this.

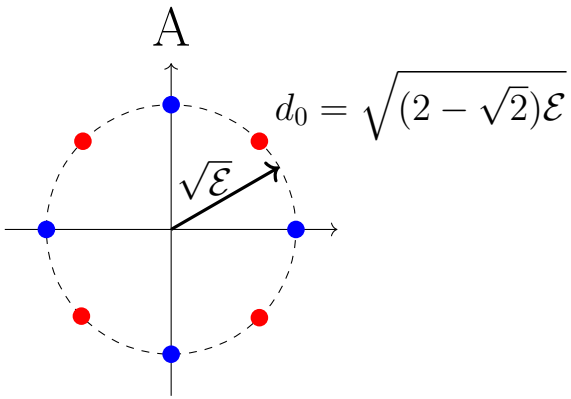
3.1 Set Partitioning

– Set partitioning involves dividing the signal constellation into subsets such that the minimum distance between points within a subset is greater than that in the original constellation.

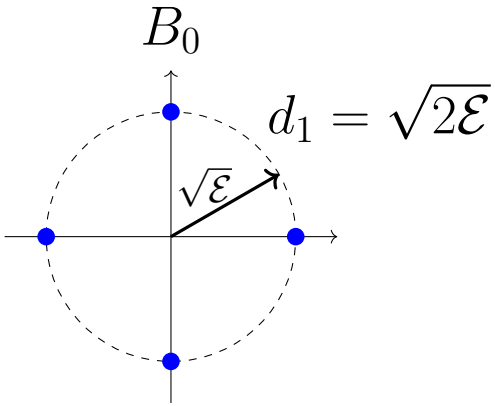
– This hierarchical partitioning improves error performance in coded modulation schemes. The following two examples illustrate the set partitioning method proposed by Ungerboeck.

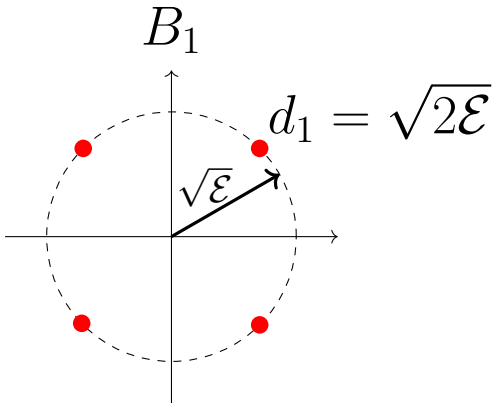
– **Example 1:** 8-PSK Constellation.

∗ The minimum distance between two points in the original 8-PSK constellation (set A) is $d_0 = \sqrt{(2-\sqrt{2})\mathcal{E}}$

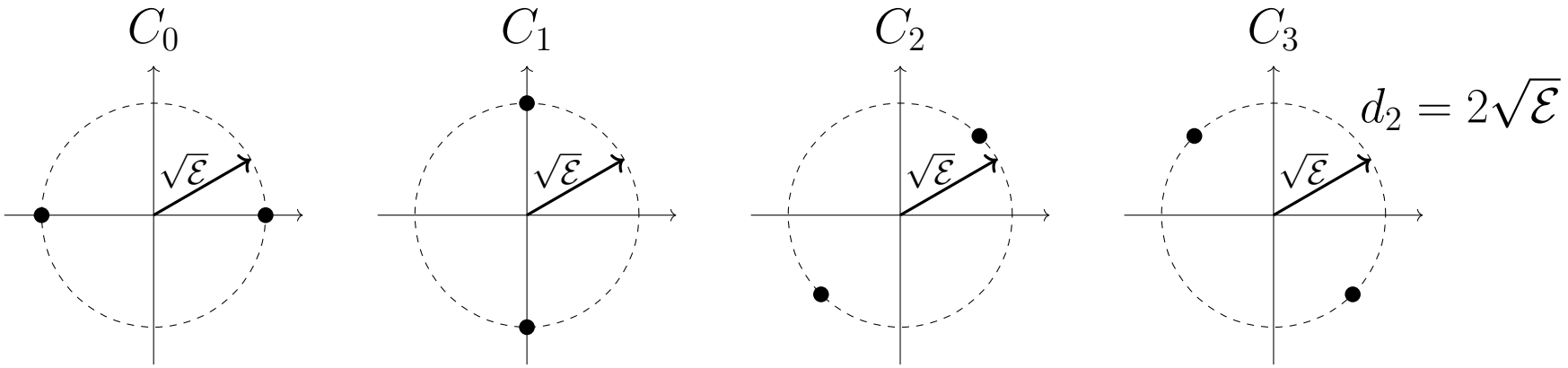


∗ Divide the constellation into two subsets, B_0 and B_1 as shown below



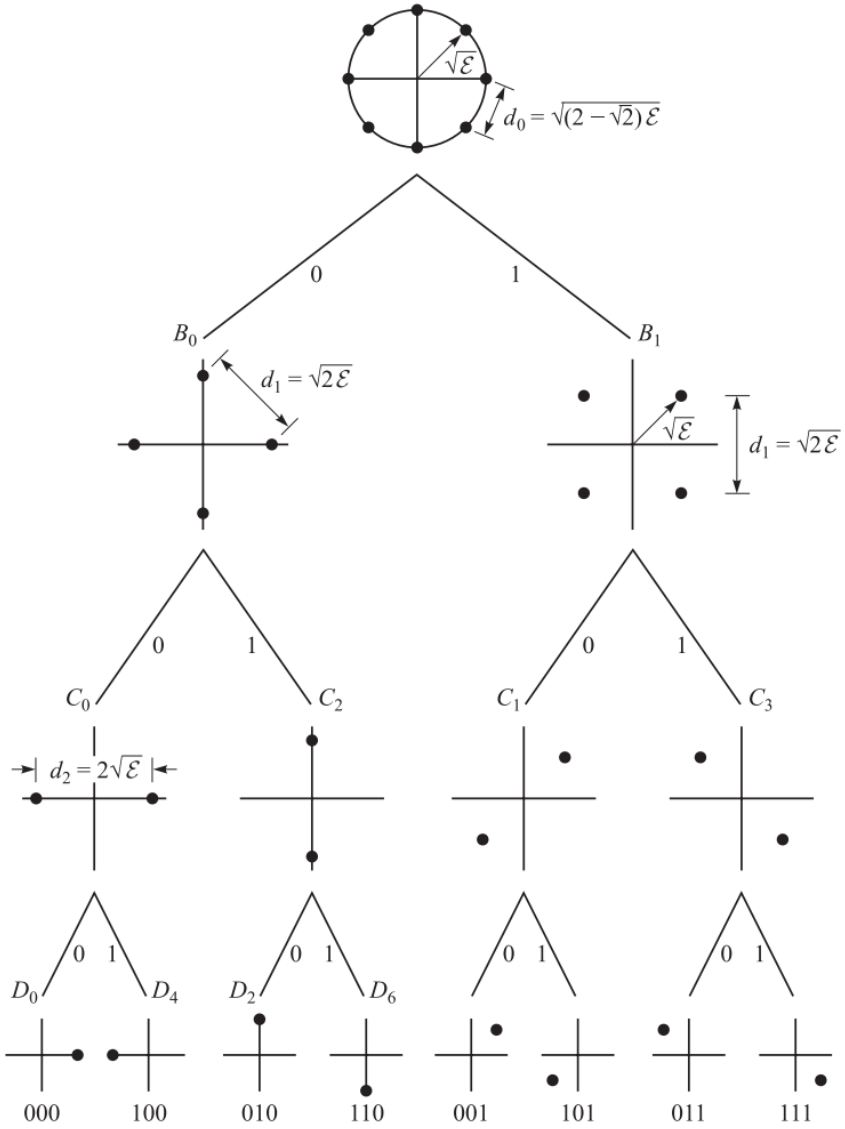


* Divide subset B_0 into two subsets, C_0 and C_1, and divide subset B_1 into two subsets, C_2 and C_3, as shown below.

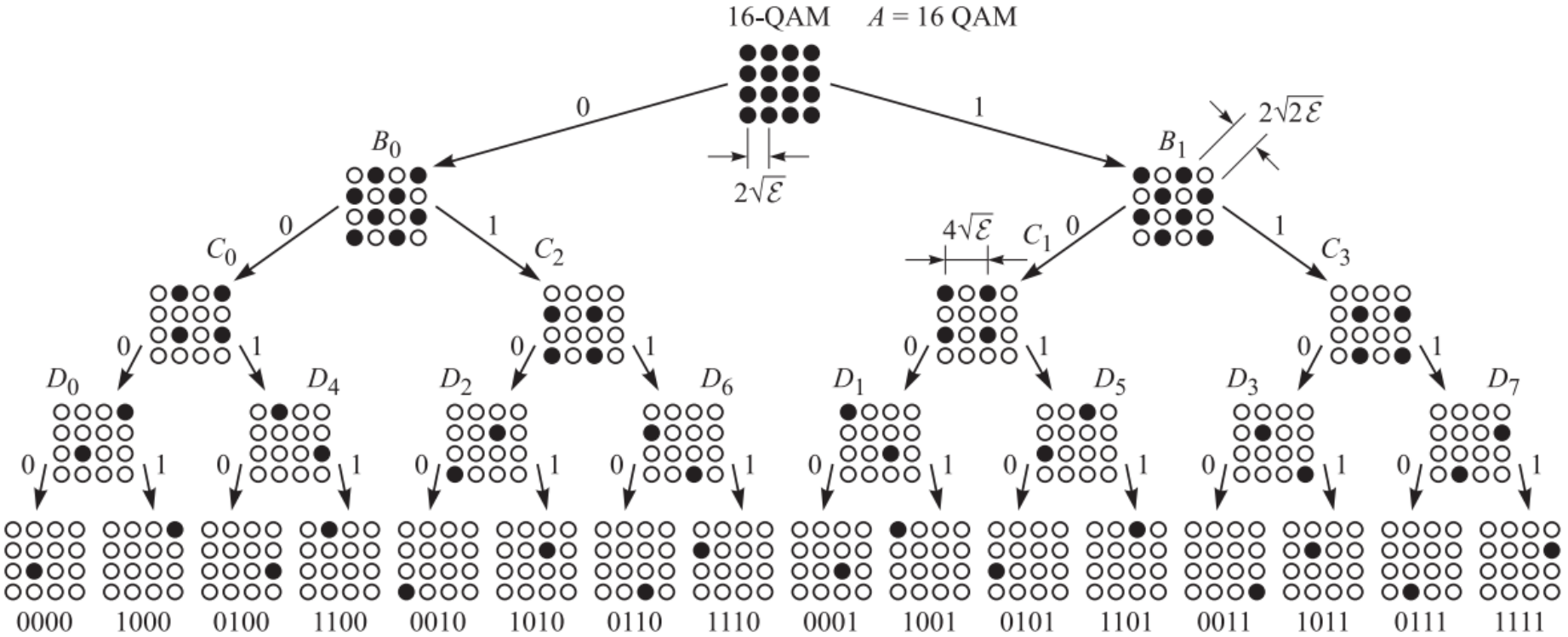


* The minimum distance between two points in subset C_i is $d_2 = 2\sqrt{\mathcal{E}}$.

– Set partitioning for 8-PSK

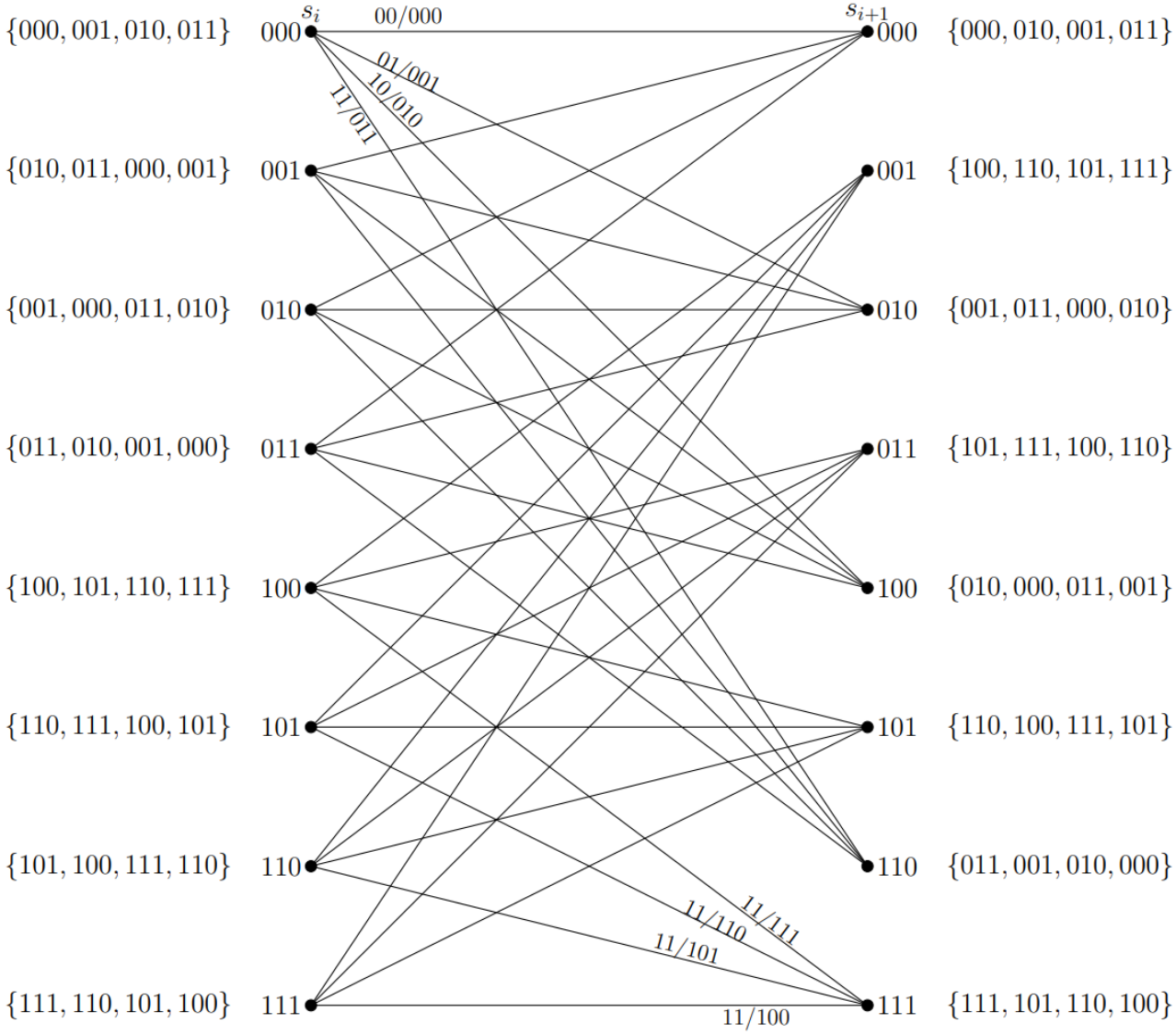


– **Example 2:** Set partitioning for 16-QAM



– For TCM to be effective, the following condition must be satisfied

> ∗ The signal points assigned to transitions that originate from or merge into the same state must belong to the same subset of the constellation.

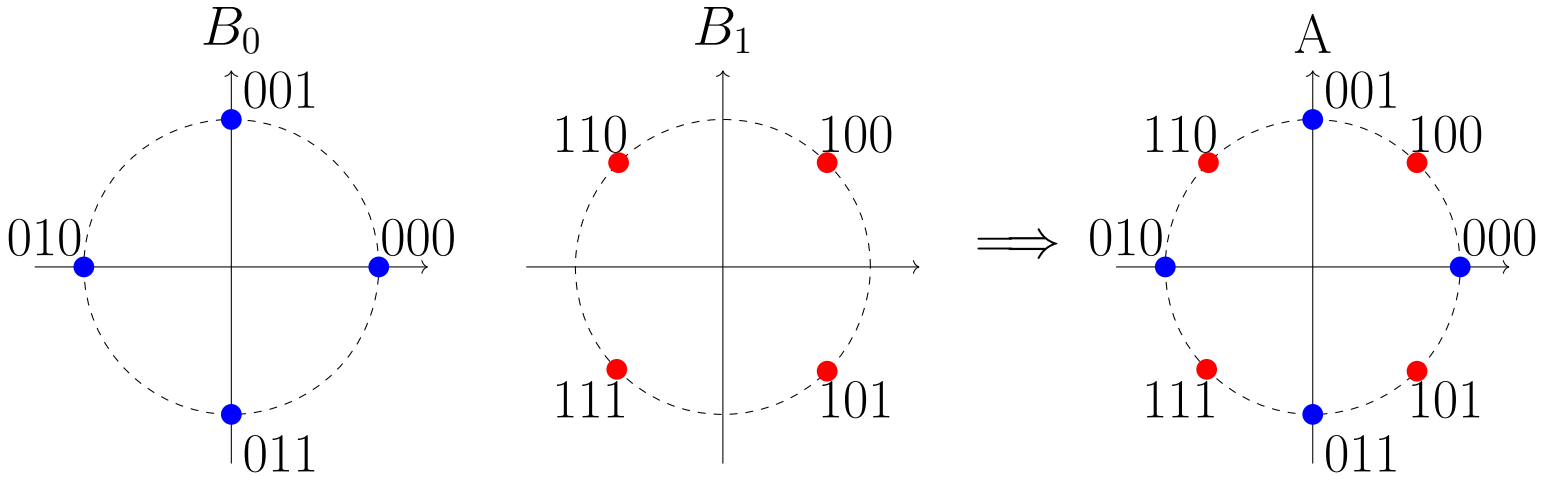

s_i
s_{i+1}
{000, 001, 010, 011}
000
00/000
01/001
10/010
11/011
{010, 011, 000, 001}
001
{001, 000, 011, 010}
010
{011, 010, 001, 000}
011
{100, 101, 110, 111}
100
{110, 111, 100, 101}
101
{101, 100, 111, 110}
110
{111, 110, 101, 100}
111
11/111
11/110
11/101
11/100
000
{000, 010, 001, 011}
001
{100, 110, 101, 111}
010
{001, 011, 000, 010}
011
{101, 111, 100, 110}
100
{010, 000, 011, 001}
101
{110, 100, 111, 101}
110
{011, 001, 010, 000}
111
{111, 101, 110, 100}

– In this example, the code symbols fall into two groups: {000, 001, 010, 011} and {100, 101, 110, 111}.

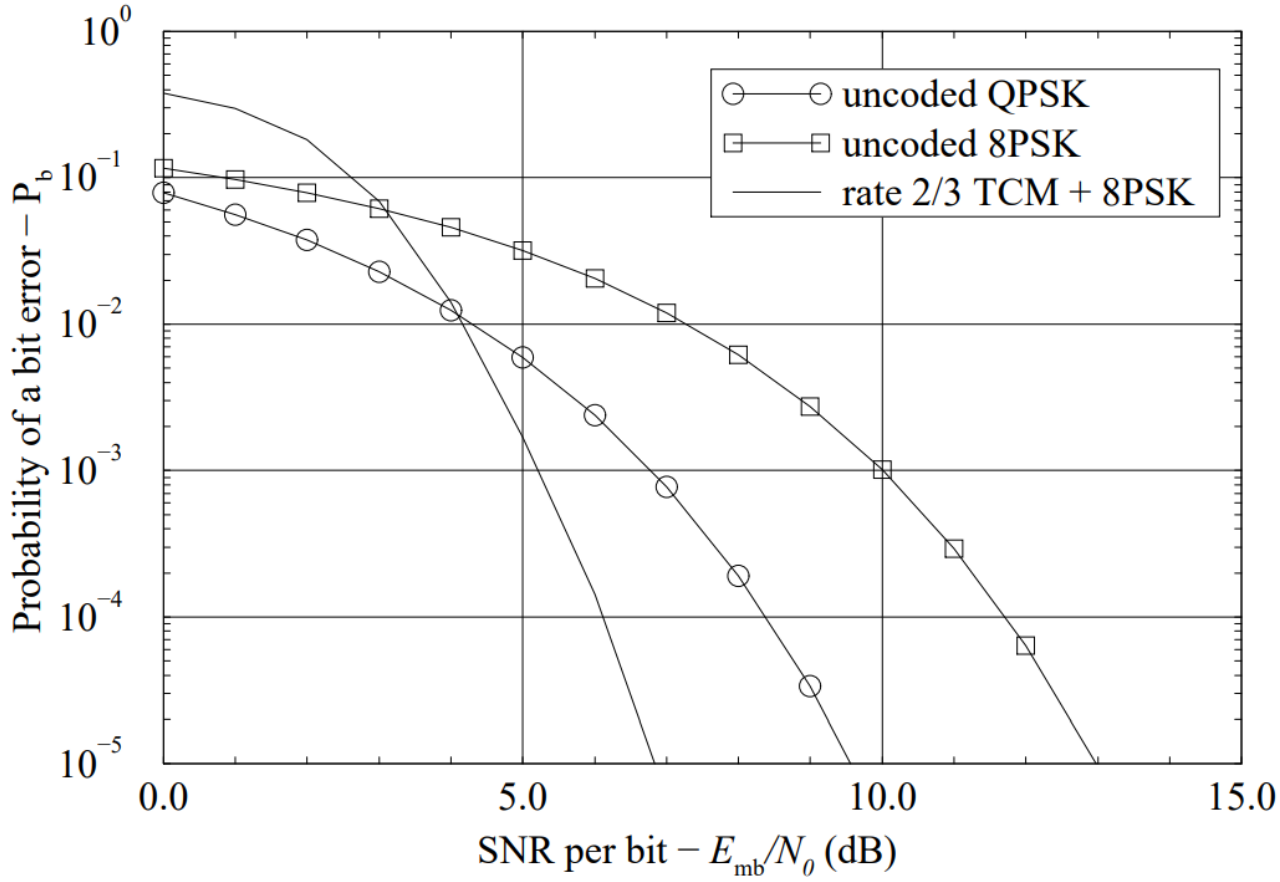
– Symbols in the first group should be mapped to signals in subset B_0, and those in the second group to subset B_1.

– **Note:** The assignment is arbitrary — the first group could also be mapped to B_1 and the second to B_0.

– One possible mapping of code symbols within each subset is



– When the above mapping of code symbols to signal points is used, the performance of the coding/modulation scheme is much better.



Why set partitioning improves error performance?

– set partitioning is a structured symbol grouping, and reduces probability of incorrect path selection by maximizing the distance between sequences that diverge and remerge in the trellis, TCM makes it less likely for the decoder to confuse them under noise.

4 Summary

- Channel coding (forward error correction) is a key technique for approaching Shannon capacity.
- We have come a long way since Shannon's seminal work, and today there are various powerful channel codes.
- Block codes process information block by block (e.g., Hamming codes).
- Convolutional codes are a class of trellis-based codes that process data continuously.
- The Viterbi algorithm provides an efficient method for decoding convolutional codes, and is optimal in that it always finds the path with the smallest path metric.
- Trellis-coded modulation (TCM) combines coding and modulation, offering significant performance gains without increasing bandwidth.

References

[1] John G. Proakis and Masoud Salehi, *Digital Communications*, McGraw-Hill, 2008.

[2] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, John Wiley & Sons, 2006.

[3] Simon Haykin, *Digital Communication Systems*, John Wiley & Sons, 2014.

[4] Tho Le-ngoc, *Digital Communications Lecture Notes*, McGill University, 2010.

[5] Hari Balakrishnan and George Verghese, *Introduction to Digital Communication Systems Lecture Notes*, MIT, 2012.

[6] Ian Marsland, *Principles of Digital Communication Lecture Notes*, Carleton University, 2017.