Principles of Communications

Meixia Tao
Shanghai Jiao Tong University

Chapter 10: Channel Coding

Topics to be Covered

- **Linear block code** (线性分组码)
- **Convolutional code** (卷积码)
Information Theory and Channel Coding

- Shannon’s noisy channel coding theorem tells that adding controlled redundancy allows transmission at arbitrarily low bit error rate (BER) as long as $R \leq C$
- Error control coding (ECC) uses this controlled redundancy to detect and correct errors
- ECC depends on the system requirements and the nature of the channel
- The key in ECC is to find a way to add redundancy to the channel so that the receiver can fully utilize that redundancy to detect and correct the errors, and to reduce the required transmit power – coding gain
Example

- We want to transmit data over a telephone link using a modem under the following conditions:
  - Link bandwidth = 3kHz
  - The modem can operate up to the speed of 3600 bits/sec at an error probability $P_e = 8 \times 10^{-4}$

- Target: transmit the data at rate of 1200 bits/sec at maximum output SNR = 13 dB with a prob. of error $10^{-4}$
Solution: Shannon Theorem

- Channel capacity is

\[ C = B \log_2 \left(1 + \frac{S}{N}\right) = 13,000 \text{ bits/sec} \]

Since \( B = 3000 \) and \( S/N = 20 \) (13 dB = \( 10\log_{10}20 \))
- Thus, by Shannon’s theorem, we can transmit the data with an arbitrarily small error probability
- Note that without coding \( P_e = 8 \times 10^{-4} \)

For the given modem, criterion \( P_e = 10^{-4} \) is not met.
Solution: A Simple Code Design

- **Repetition code:** every bit is transmitted 3 times when $b_k = "0"$ or "$1"$, transmit codeword "$000" or "$111".

- Based on the received codewords, the decoder attempts to extract the transmitted bits using **majority-logic decoding scheme**.

- Clearly, the transmitted bits will be recovered correctly as long as no more than one of the bits in the codeword is affected by noise.

<table>
<thead>
<tr>
<th>Tx bits $b_k$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codewords</td>
<td>000 000 000 000 111 111 111 111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rx bits</td>
<td>000 001 010 100 011 101 110 111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tilde{b}_k$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
With this simple error control coding, the probability of error is

\[ P_e = P(b_k \neq \hat{b}_k) \]

\[ = P(2 \text{ or more bits in codeword are in error}) \]

\[ = \binom{3}{2} q_c^2 (1 - q_c) + \binom{3}{3} q_c^3 \]

\[ = 3q_c^2 - 2q_c^3 \]

\[ = 0.0192 \times 10^{-4} \]

\[ \leq \text{Required } P_e \text{ of } 10^{-4} \]
Channel Coding

- Coding techniques are classified as either block codes or convolutional codes, depending on the presence or absence of memory.

- A block code has no memory:
  - Information sequence is broken into blocks of length $k$.
  - Each block of $k$ information bits is encoded into a block of $n$ coded bits.
  - No memory from one block to another block.

- A convolutional code has memory:
  - A shift register of length $k_0 L$ is used.
  - Information bits enter the shift register $k_0$ bits at a time; then $n_0$ coded bits are generated.
  - These $n_0$ bits depend not only on the recent $k_0$ bit that just entered the shift register, but also on the $k_0(L - 1)$ previous bits.
Block Codes

- An \((n,k)\) block code is a collection of \(M = 2^k\) codewords of length \(n\).
- Each codeword has a block of \(k\) information bits followed by a group of \(r = n-k\) check bits that are derived from the \(k\) information bits in the block preceding the check bits.
- The code is said to be linear if any linear combination of 2 codewords is also a codeword.
  - i.e. if \(C_i\) and \(C_j\) are codewords, then \(C_i + C_j\) is also a codeword (where the addition is always module-2).
- Code rate (rate efficiency) = $\frac{k}{n}$

- Matrix description
  - codeword $\mathbf{c} = (c_1, c_2, \ldots, c_n)$
  - message bits $\mathbf{m} = (m_1, m_2, \ldots, m_k)$

- Each block code can be generated using a **Generator Matrix** $\mathbf{G}$ (dim: $k \times n$)

- Given $\mathbf{G}$, then
  
\[ \mathbf{c} = \mathbf{mG} \]

Codeword

message
Generator Matrix $G$

$$G = [I_k | P]_{k \times n}$$

\[
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
p_{11} & p_{12} & \cdots & p_{1,n-k} \\
p_{21} & p_{22} & \cdots & p_{2,n-k} \\
\vdots & \vdots & \ddots & \vdots \\
p_{k,1} & p_{k,2} & \cdots & p_{k,n-k}
\end{bmatrix}
\]

- $I_k$ is identity matrix of order $k$
- $P$ is matrix of order $k \times (n-k)$, which is selected so that the code will have certain desirable properties.
Systematic Codes

- The form of $G$ implies that the $1^{st}$ $k$ components of any codeword are precisely the information symbols.
- This form of linear encoding is called **systematic encoding**.
- Systematic-form codes allow easy implementation and quick look-up features for decoding.
- For linear codes, any code is equivalent to a code in **systematic form** (given the same performance). Thus we can restrict our study to only systematic codes.
Example: Hamming Code

- A family of (n,k) linear block codes that have the following parameters:
  - Codeword length \( n = 2^m - 1, \quad m \geq 3 \)
  - # of message bits \( k = 2^m - m - 1 \)
  - # of parity check bits \( n - k = m \)
  - Capable of providing single-error correction capability with \( d_{\text{min}} = 3 \)
(7, 4) Hamming Code

- Consider a (7,4) Hamming code with generator matrix

\[
G = \begin{bmatrix}
1 & 0 & 0 & 0 & | & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & | & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & | & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & | & 1 & 0 & 1 \\
\end{bmatrix}
\]

- Find all codewords
Solution

- Let \( m = [1 \ 1 \ 1 \ 1] \)

\[
c = mG = [1 \ 1 \ 1 \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}
\]

\[
= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
\]
### List of all Codewords

- $n = 7$, $k = 4 \rightarrow 2^k = 16$ message blocks

<table>
<thead>
<tr>
<th>Message</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>0 0 0 1 1 0 0 1 0</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>0 0 1 0 1 0 1 1 1</td>
</tr>
<tr>
<td>0 0 1 1 1</td>
<td>0 0 1 1 0 0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>0 1 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 1 1</td>
<td>0 1 0 0 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>0 1 1 0 0 1 1 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1</td>
<td>0 1 1 1 1 0 0 0 1</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 1 1 0 0</td>
</tr>
<tr>
<td>1 0 0 1 1</td>
<td>1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>1 0 1 0 0 0 0 0 1</td>
</tr>
<tr>
<td>1 0 1 1 1</td>
<td>1 0 1 1 1 0 0 0 1</td>
</tr>
<tr>
<td>1 1 0 0 0</td>
<td>1 1 0 0 0 1 1 0 0</td>
</tr>
<tr>
<td>1 1 0 1 1</td>
<td>1 1 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>1 1 1 0 0 1 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
Parity Check Matrix

- For each $G$, it is possible to find a corresponding parity check matrix $H$
  \[ H = \begin{bmatrix} P^T & I_{n-k} \end{bmatrix} (n-k) \times n \]

- $H$ can be used to verify if a codeword $C$ is generated by $G$

- Let $C$ be a codeword generated by $G = [I_k | P]_{k \times n}$

\[ cH^T = mGH^T = 0 \]

**Example**: Find the parity check matrix of (7,4) Hamming code
Error Syndrome

- Received codeword \( r = c + e \)
  where \( e = \text{Error vector or Error Pattern} \)
  it is 1 in every position where data word is in error

- Example

  \[
  \begin{align*}
  c &= [1 \ 0 \ 1 \ 0] \\
  r &= [1 \ 1 \ 0 \ 0] \\
  e &= [0 \ 1 \ 1 \ 0]
  \end{align*}
  \]
Error Syndrome (cont’d)

- \( s \triangleq rH^T = \text{Error Syndrome} \)
- But

\[
s = rH^T = (c + e)H^T \\
= cH^T + eH^T \\
= eH^T
\]

1. If \( s=0 \rightarrow r = c \) and \( m \) is the 1\(^{\text{st}}\) \( k \) bits of \( r \)

2. If \( s \neq 0 \), and \( s \) is the \( j^{\text{th}} \) row of \( H^T \rightarrow 1 \) error in \( j^{\text{th}} \) position of \( r \)
Consider the (7,4) Hamming code example

\[ H^T = [P^T | I_{n-k}]^T = \begin{bmatrix} P \\ I_{n-k} \end{bmatrix} \]

\[ = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

How many error syndromes in total

- Note that \( s \) is the last row of \( H^T \)
- Also note error took place in the last bit

\[ \Rightarrow \text{ Syndrome indicates position of error } \]

So if \( r = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \)

\[ \Rightarrow rH^T = [0 \ 0 \ 0] \]

But if \( r = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0] \)

\[ \Rightarrow rH^T = [0 \ 0 \ 1] = \text{Error syndrome } s \]
Cyclic Codes

- A code \( C = \{c_1, c_2, \ldots, c_{2^k}\} \) is cyclic if

\[
(c_1, c_2, \ldots, c_n) \in C \quad \Rightarrow \quad (c_n, c_1, \ldots, c_{n-1}) \in C
\]

- \((7,4)\) Hamming code is cyclic

<table>
<thead>
<tr>
<th>message</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0001101</td>
</tr>
<tr>
<td>1000</td>
<td>1000110</td>
</tr>
<tr>
<td>0100</td>
<td>0100011</td>
</tr>
</tbody>
</table>
Important Parameters

- **Hamming Distance** between codewords $c_i$ and $c_j$:
  \[ d(c_i, c_j) = \text{# of components at which the 2 codewords differ} \]

- **Hamming weight** of a codeword $c_i$ is
  \[ w(c_i) = \text{# of non-zero components in the codeword} \]

- **Minimum Hamming Distance** of a code:
  \[ d_{\text{min}} = \min d(c_i, c_j) \text{ for all } i \neq j \]

- **Minimum Weight** of a code:
  \[ w_{\text{min}} = \min w(c_i) \text{ for all } c_i \neq 0 \]

- **Theorem**: In any linear code, $d_{\text{min}} = w_{\text{min}}$

- **Exercise**: Find $d_{\text{min}}$ for $(7,4)$ Hamming code
Soft-Decision and Hard-Decision Decoding

- **Soft-decision decoder** operates directly on the decision statistics

- **Hard-decision decoder** makes “hard” decision (0 or 1) on individual bits

- Here we only focus on hard-decision decoder
Hard-Decision Decoding

- Minimum Hamming Distance Decoding
  - Given the received codeword $r$, choose $c$ which is closest to $r$ in terms of Hamming distance.
  - To do so, one can do an exhaustive search – too much if $k$ is large.

- Syndrome Decoding
  - Syndrome testing: $r = c + e$ with $s = rH^T$
  - This implies that the corrupted codeword $r$ and the error pattern have the same syndrome.
  - A simplified decoding procedure based on the above observation can be used.
Standard Array

- Let the codewords be denoted as \( \{c_1, c_2, \ldots, c_M\} \) with \( c_1 \) being the all-zero codeword.

- A standard array is constructed as:

\[
\begin{array}{ccc}
\{c_1, c_2, \ldots, c_M\} \\
\{e_1, e_1 \oplus c_2, \ldots, e_1 \oplus c_M\} \\
\{e_2, e_2 \oplus c_2, \ldots, e_2 \oplus c_M\} \\
\vdots \\
\{e_{2^{n-k}-1}, e_{2^{n-k}-1} \oplus c_2, \ldots, e_{2^{n-k}-1} \oplus c_M\} \\
\end{array}
\]

Syndrome \( s \)

\[
\begin{align*}
0 \\
\text{s} = e_1 H^T \\
\text{s} = e_2 H^T \\
\text{s} = e_{2^{n-k}-1} H^T
\end{align*}
\]
Hard-Decoding Procedure

- Find the syndrome by $r$ using $s = rH^T$
- Find the coset corresponding to $s$ by using the standard array
- Find the cost leader and decode as $c = r + e_j$
- Exercise: try (7,4) Hamming code
Error Correction Capability

- A linear block code with a minimum distance $d_{\text{min}}$ can
  - Detect up to $(d_{\text{min}} - 1)$ errors in each codeword
  - Correct up to $t = \left\lfloor \frac{d_{\text{min}} - 1}{2} \right\rfloor$ errors in each codeword
  - $t$ is known as the error correction capability of the code

\[
d(c_i, c_j) \geq 2t + 1 \quad \text{and} \quad d(c_i, c_j) < 2t
\]
Probability of Codeword Error for Hard-Decision Decoding

- Consider a linear block code (n, k) with an error correcting capability \( t \). The decoder can correct all combination of errors up to and including \( t \) errors.

- Assume that the error probability of each individual coded bit is \( p \) and that bit errors occur independently since the channel is memoryless.

- If we send \( n \)-bit block, the probability of receiving a specific pattern of \( m \) errors and \( (n-m) \) correct bits is

\[
p^m(1 - p)^{n-m}
\]
- Total number of distinct pattern of $n$ bits with $m$ errors and $(n-m)$ correct bits is

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

- Total probability of receiving a pattern with $m$ errors is

$$P(m, n) = \binom{n}{m} \cdot p^m (1-p)^{n-m}$$

- Thus, the codeword error probability is upper-bounded by

$$P_M \leq \sum_{m=t+1}^{n} \binom{n}{m} p^m (1-p)^{n-m}$$

(with equality for perfect codes)
Error Detection vs. Error Correction

- To detect $e$ bit errors, we have $d_{\min} \geq e + 1$
- To correct $t$ bit errors, we have $d_{\min} \geq 2t + 1$
Major Classes of Block Codes

- Repetition Code
- Hamming Code
- Golay Code
- BCH Code
- Reed-Solomon Codes
- Walsh Codes

- LDPC Codes: invented by Robert Gallager in his PhD thesis in 1960, now proved to be capacity-approaching
Convolutional Codes

- A convolutional code has memory
  - It is described by 3 integers: n, k, and L
  - Maps k bits into n bits using previous (L-1) k bits
  - The n bits emitted by the encoder are not only a function of the current input k bits, but also a function of the previous (L-1)k bits
  - k/n = Code Rate (information bits/coded bit)
  - L is the constraint length and is a measure of the code memory
  - n does not define a block or codeword length
Convolutional Encoding

- A rate k/n convolutional encoder with constraint length L consists of
  - kL-stage shift register and n mod-2 adders
- At each unit of time:
  - k bits are shifted into the 1st k stages of the register
  - All bits in the register are shifted k stages to the right
  - The output of the n adders are sequentially sampled to give the coded bits
  - There are n coded bits for each input group of k information or message bits. Hence $R = \frac{k}{n}$ information bits/coded bit is the code rate ($k < n$)
Encoder Structure
(rate $k/n$, constraint length $L$)

- Typically, $k=1$ for binary codes. Hence, consider rate $1/n$ codes
Convolution Codes Representation

- **Encoding function**: characterizes the relationship between the information sequence \( m \) and the output coded sequence \( U \)

- Four popular methods for representation
  - Connection pictorial and connection polynomials (usually for encoder)
  - State diagram
  - Tree diagram
  - Trellis diagram (Usually for decoder)
Connection Representation

- Specify \( n \) connection vectors, \( g_i, i = 1, \ldots, n \) for each of the \( n \) mod-2 adders

- Each vector has \( kL \) dimension and describes the connection of the shift register to the mod-2 adders

- A 1 in the \( i^{th} \) position of the connection vector implies shift register is connected

- A 0 implies no connection exists
Example: $L = 3$, Rate $1/2$

If Initial Register content is $0 \ 0 \ 0$ and Input Sequence is $1 \ 0 \ 0$. Then Output Sequence is $11 \ 10 \ 11$

Or

$g_1 = 111$
$g_2 = 101$

$g_1(X) = 1 + X + X^2$
$g_2(X) = 1 + X^2$

If Initial Register content is $0 \ 0 \ 0$ and Input Sequence is $1 \ 0 \ 0$. Then Output Sequence is $11 \ 10 \ 11$
State Diagram Representation

- The contents of the rightmost L-1 stages (or the previous L-1 bits) are considered the current state $\Rightarrow 2^{L-1}$ states

- Knowledge of the current state and the next input is necessary and sufficient to determine the next output and next state

- For each state, there are only 2 transitions (to the next state) corresponding to the 2 possible input bits

- The transitions are represented by paths on which we write the output word associated with the state transition
  - A solid line path corresponds to an input bit 0
  - A dashed line path corresponds to an input bit 1
Example: $L = 3$, Rate = $1/2$

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Assume that $m = 11011$ is the input followed by $L-1 = 2$ zeros to flush the register. Also assume that the initial register contents are all zero. Find the output sequence $U$

<table>
<thead>
<tr>
<th>Input bit $m_i$</th>
<th>Register contents</th>
<th>State at time $t_i$</th>
<th>State at time $t_{i+1}$</th>
<th>Branch word at time $t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>000</td>
<td>00</td>
<td>00</td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>00</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>10</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>11</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>01</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>10</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>11</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>01</td>
<td>00</td>
<td>0</td>
</tr>
</tbody>
</table>

Output sequence: $U = 11 01 01 00 01 01 11$
Trellis Diagram

- The trellis diagram is similar to the state diagram, except that it adds the dimension of time.
- The code is represented by a trellis where each trellis branch describes an output word.

Blue trellis lights

-- Columbus Park, New York City
Trellis Diagram

Trellis structure repeats itself after depth $L = 3$
Every input sequence \((m_1, m_2, \ldots)\) corresponds to

- a path in the trellis
- a state transition sequence \((s_0, s_1, \ldots)\), (assume \(s_0 = 0\) is fixed)
- an output sequence \(((u_1, u_2), (u_3, u_4), \ldots)\)

Example: Let \(s_0 = 00\), then

- \(b_1 b_2 b_3 = 000\) gives output \(000000\) and states \(aaaa\)
- \(b_1 b_2 b_3 = 100\) gives output \(111011\) and states \(abca\)

\[
\begin{array}{cccccc}
\text{a} & = & 00 & \rightarrow & 00 & \rightarrow & 00 & \rightarrow & 00 & \rightarrow & 00 \\
\text{b} & = & 10 & \rightarrow & 11 & \rightarrow & 10 & \rightarrow & 10 & \rightarrow & 11 & \rightarrow & 10 \\
\text{c} & = & 01 & \rightarrow & 01 & \rightarrow & 10 & \rightarrow & 01 & \rightarrow & 01 \\
\text{d} & = & 11 & \rightarrow & 11 & \rightarrow & 11 & \rightarrow & 11 & \rightarrow & 11 \\
\end{array}
\]
- We have introduced conv. code
  - Constraint length $L$ and rate $R = 1/n$
  - Polynomials representation
  - State diagram representation
  - Trellis diagram representation

- We will talk about decoding of convolutional code
  - Maximum Likelihood Decoding
  - Viterbi Algorithm
  - Transfer Function
Maximum Likelihood Decoding

- Transmit a coded sequence $U^{(m)}$ (correspond to message sequence $m$) using a digital modulation scheme (e.g. BPSK or QPSK)
- Received sequence $z$
- Maximum likelihood decoder
  - Find the sequence $U^{(j)}$ such that
    $$P(Z|U^j) = \max_{U^{(m)}} P(Z|U^{(m)})$$
  - Will minimize the probability of error if $m$ is equally likely
Maximum Likelihood Metric

- Assume a memoryless channel, i.e. noise components are independent. Then, for a rate 1/n code

\[ P(Z|U^{(m)}) = \prod_{i=1}^{\infty} P(Z_i|U_i^{(m)}) = \prod_{i=1}^{\infty} \prod_{j=1}^{n} P(z_{ji}|u_{ji}^{(m)}) \]

- Then the problem is to find a path through the trellis such that

\[ \max_{U^{(m)}} \prod_{i=1}^{\infty} \prod_{j=1}^{n} P(z_{ji}|u_{ji}^{(m)}) \]

by taking log

\[ \max_{U^{(m)}} \sum_{i=1}^{\infty} \sum_{j=1}^{n} \log P(z_{ji}|u_{ji}^{(m)}) \]

\[ = \max_{U^{(m)}} \sum_{i=1}^{\infty} \sum_{j=1}^{n} LL\left(z_{ji}|u_{ji}^{(m)}\right) \]
Decoding Algorithm: Log-Likelihood

- For AWGN channel (soft-decision)
  - \( z_{ji} = u_{ji} + n_{ji} \) and \( P(z_{ji}|u_{ji}) \) is Gaussian with mean \( u_{ji} \) and variance \( \sigma^2 \)
  - Hence
    \[
    \ln p(z_{ji}|u_{ji}) = -\frac{1}{2} \ln (2\pi\sigma^2) - \frac{(z_{ji} - u_{ji})^2}{2\sigma^2}
    \]
  - Note that the objective is to compare which \( \Sigma_i \ln(p(z|u)) \) for different \( u \) is larger, hence, constant and scaling does not affect the results
  - Then, we let the log-likelihood be
    \[
    LL(z_{ji}|u_{ji}) = -(z_{ji} - u_{ji})^2
    \]
    and
    \[
    \ln P(Z|U^{(m)}) = -\sum_{i=1}^{\infty} \sum_{j=1}^{n} (z_{ji} - u_{ji}^{(m)})^2
    \]
  - Thus, soft decision ML decoder is to choose the path whose corresponding sequence is at the minimum Euclidean distance to the received sequence
For binary symmetric channel (hard decision)

\[
\begin{align*}
    p(z | u) &= \begin{cases} 
    p & \text{if } z \neq u \\
    1 - p & \text{if } z = u 
    \end{cases} \\

    LL(z_{ji} | u_{ji}) &= \ln p(z_{ji} | u_{ji}) = \begin{cases} 
    \ln p & \text{if } z_{ji} \neq u_{ji} \\
    \ln(1 - p) & \text{if } z_{ji} = u_{ji} 
    \end{cases} \\
    &= \begin{cases} 
    \ln p/(1 - p) & \text{if } z_{ji} \neq u_{ji} \\
    0 & \text{if } z_{ji} = u_{ji} 
    \end{cases} \\
    &= \begin{cases} 
    -1 & \text{if } z_{ji} \neq u_{ji} \\
    0 & \text{if } z_{ji} = u_{ji} 
    \end{cases} \\
\end{align*}
\]

(since p<0.5)

Thus

\[
    \log P(Z | U^{(m)}) = -d_m
\]

Hard-Decision ML Decoder = Minimum Hamming Distance Decoder
Maximum Likelihood Decoding Procedure

- **Compute**, for each branch $i$, the branch metric using the output bits $\{u_{1,i}, u_{2,i}, \ldots, u_{n,i}\}$ associated with that branch and the received symbols $\{z_{1,i}, z_{2,i}, \ldots, z_{n,i}\}$

- **Compute**, for each valid path through the trellis (a valid codeword sequence $U^{(m)}$), the sum of the branch metrics along that path

- The path with the maximum path metric is the decoded path

- To compare all possible valid paths we need to do exhaustive search or brute-force, not practical as the # of paths grow exponentially as the path length increases

- The optimum algorithm for solving this problem is the Viterbi decoding algorithm or Viterbi decoder
Andrew Viterbi
(1935- )

- BS & MS in MIT
- PhD in University of Southern California
- Invention of Viterbi algorithm in 1967
- Co-founder of Qualcomm Inc. in 1983
Viterbi Decoding (R=1/2, L=3)

Input data sequence $\mathbf{m}$: 1 1 0 1 1 1 ...

Coded sequence $\mathbf{U}$: 11 0 1 01 00 01 ...

Received sequence $\mathbf{Z}$: 11 0 1 01 10 01 ...

Branch metric

- $a=00$
- $b=10$
- $c=01$
- $d=11$
Viterbi Decoder

- Basic idea:
  - If any 2 paths in the trellis merge to a single state, one of them can always be eliminated in the search

- Let cumulative path metric of a given path at $t_i = \text{sum of the branch metrics along that path up to time } t_i$

- Consider $t_5$
  - The upper path metric is 4, the lower math metric is 1
  - The upper path metric CANNOT be part of the optimum path since the lower path has a lower metric
  - This is because future output branches depend only on the current state and not the previous state
Path Metrics for 2 Merging Paths

Path metric = 4

Path metric = 1
Viterbi Decoding

- At time $t_i$, there are $2^{L-1}$ states in the trellis.
- Each state can be entered by means of 2 states.
- Viterbi Decoding consists of computing the metrics for the 2 paths entering each state and eliminating one of them.
- This is done for each of the $2^{L-1}$ nodes at time $t_i$.
- The decoder then moves to time $t_{i+1}$ and repeats the process.
Example
**Distance Properties**

- \( d_{\text{free}} = \text{Minimum Free distance} = \text{Minimum distance of any pair of arbitrarily long paths that diverge and remerge} \)

- A code can correct any \( t \) channel errors where (this is an approximation)
  \[
  t \leq \left\lfloor \frac{d_{\text{free}} - 1}{2} \right\rfloor
  \]
Transfer Function

- The distance properties and the error rate performance of a convolutional code can be obtained from its transfer function.

- Since a convolutional code is linear, the set of Hamming distances of the code sequences generated up to some stages in the trellis, from the all-zero code sequence, is the same as the set of distances of the code sequences with respect to any other code sequence.

- Thus, we assume that the all-zero path is the input to the encoder.
State Diagram Labeled according to distance from all-zero path

- $D^m$ denote $m$ non-zero output bits
- $N$ if the input bit is non-zero
- $L$ denote a branch in the path

\[ X_b = D^2 NLX_a + LNX_c \]
\[ X_c = DLX_b + DLX_d \]
\[ X_d = DNLX_b + DNLX_d \]
\[ X_e = D^2 LX_c \]
The transfer function $T(D, N, L)$, also called the weight enumerating function of the code is

$$T(D, N, L) = \frac{X_e}{X_a}$$

By solving the state equations we get

$$T(D, N, L) = \frac{D^5NL^3}{1 - DNL(1 + L)}$$

$$= D^5NL^3 + D^6N^2L^4(1 + L) + D^7N^3L^5(1 + L)^2$$

$$+ \ldots + D^{l+5}N^{l+1}L^{l+3}(1 + L)^l + \ldots$$

The transfer function indicates that:

- There is one path at distance 5 and length 3, which differs in 1 input bit from the correct all-zeros path
- There are 2 paths at distance 6, one of which is of length 4, the other length 5, and both differ in 2 input bits from all-zero path
- $d_{\text{free}} = 5$
Known Good Convolutional Codes

- Good convolutional codes can only be found in general by computer search.
- There are listed in tables and classified by their constraint length, code rate, and their generator polynomials or vectors (typically using octal notation).
- The error-correction capability of a convolutional code increases as $n$ increases or as the code rate decreases.
- Thus, the channel bandwidth and decoder complexity increases.
# Good Codes with Rate 1/2

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generator Polynomials</th>
<th>$d_{\text{free}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5,7)</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>(15,17)</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>(23,35)</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>(53,75)</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>(133,171)</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>(247,371)</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>(561,753)</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>(1167,1545)</td>
<td>12</td>
</tr>
</tbody>
</table>
## Good Codes with Rate 1/3

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generator Polynomials</th>
<th>$d_{\text{free}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5,7,7)</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>(13,15,17)</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>(25,33,37)</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>(47,53,75)</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>(133,145,175)</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>(225,331,367)</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>(557,663,711)</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>(1117,1365,1633)</td>
<td>20</td>
</tr>
</tbody>
</table>
Basic Channel Coding for Wideband CDMA

Convolutional Codes

BER = 10^{-3}

Block Codes

BER = 10^{-6}

Outer coding (RS) → Outer interleaving → Inner coding (conv.) → Inner interleaving

Service-specific coding

Convolutional code is rate 1/3 and rate 1/2, all with constraint length 9
Channel Coding for Wireless LAN (IEEE802.11a)

Input bits → Conv. Encoder \( r=1/2, K=7 \) → Puncturing → Baseband Modulator → OFDM → TX signals

<table>
<thead>
<tr>
<th>Speed (Mbps)</th>
<th>Modulation and coding rate (R)</th>
<th>Coded bits per carrier(^a)</th>
<th>Coded bits per symbol</th>
<th>Data bits per symbol(^b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>BPSK, R=1/2</td>
<td>1</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>BPSK, R=3/4</td>
<td>1</td>
<td>48</td>
<td>36</td>
</tr>
<tr>
<td>12</td>
<td>QPSK, R=1/2</td>
<td>2</td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>18</td>
<td>QPSK, R=3/4</td>
<td>2</td>
<td>96</td>
<td>72</td>
</tr>
<tr>
<td>24</td>
<td>16-QAM, R=1/2</td>
<td>4</td>
<td>192</td>
<td>96</td>
</tr>
<tr>
<td>36</td>
<td>16-QAM, R=3/4</td>
<td>4</td>
<td>192</td>
<td>144</td>
</tr>
<tr>
<td>48</td>
<td>64-QAM, R=2/3</td>
<td>6</td>
<td>288</td>
<td>192</td>
</tr>
<tr>
<td>54</td>
<td>64-QAM, R=3/4</td>
<td>6</td>
<td>288</td>
<td>216</td>
</tr>
</tbody>
</table>

Source: 802.11 Wireless Networks: The Definitive Guide / by M. Gast / O’Reilly
Other Advanced Channel Coding

- Low-density parity check codes: Robert Gallager 1960
- Turbo codes: Berrou et al 1993
- Trellis-coded modulation: Ungerboeck 1982
- Space-time coding: Vahid Tarokh et al 1998
- Polar codes: Erdal Arikan 2009
Exercise

- Find out the coding techniques adopted in LTE