Part 2.3 Convolutional codes
Overview of Convolutional Codes (1)

Convolutional codes offer an approach to error control coding substantially different from that of block codes.

- A convolutional encoder:
  - encodes the entire data stream, into a single codeword.
  - maps information to code bits sequentially by convolving a sequence of information bits with “generator” sequences.
  - does not need to segment the data stream into blocks of fixed size (Convolutional codes are often forced to block structure by periodic truncation).
  - is a machine with memory.

- This fundamental difference imparts a different nature to the design and evaluation of the code.
  - Block codes are based on algebraic/combinatorial techniques.
  - Convolutional codes are based on construction techniques.
    - Easy implementation using a linear finite-state shift register
Overview of Convolutional Codes (2)

- A convolutional code is specified by *three parameters* \( (n, k, K) \) or \( (k/n, K) \) where
  - \( k \) inputs and \( n \) outputs
    - In practice, usually \( k=1 \) is chosen.
  - \( R_c = k/n \) is the coding rate, determining the number of data bits per coded bit.
  - \( K \) is *the constraint length* of the convolutional code (where the encoder has \( K-1 \) memory elements).
Overview of Convolutional Codes (3)

Convolutional encoder
Overview of Convolutional Codes (4)

The performance of a convolutional code depends on the coding rate and the constraint length.

- **Longer constraint length $K$**
  - More powerful code
  - More coding gain
    - Coding gain: the measure in *the difference between the signal to noise ratio (SNR) levels* between the uncoded system and coded system required to reach the same bit error rate (BER) level
  - More complex decoder
  - More decoding delay

- **Smaller coding rate $R_c = k/n$**
  - More powerful code due to extra redundancy
  - Less bandwidth efficiency
Overview of Convolutional Codes (5)

![Graphs showing coding gain vs constraint length for different rates and constraint lengths.]

Figure 8.52 Dependence of realizable coding gains on constraint length for rate $\frac{1}{2}$ convolutional codes.

Figure 8.53 Dependence of realizable coding gains on constraint length for rate $\frac{1}{3}$ convolutional codes.
An Example of Convolutional Codes (1)

Convolutional encoder (rate $\frac{1}{2}$, K=3)

- 3 shift-registers, where the first one takes the incoming data bit and the rest form the memory of the encoder.
An Example of Convolutional Codes (2)

Message sequence: \( m = (101) \)
An Example of Convolutional Codes (3)

\[ m = (101) \xrightarrow{\text{Encoder}} U = (11 10 00 10 11) \]

\[ R_{eff} = \frac{3}{10} < R_c = \frac{1}{2} \]
Initialize the memory before encoding the first bit (all-zero)
Clear out the memory after encoding the last bit (all-zero)
  - Hence, a tail of zero-bits is appended to data bits.

**Effective code rate:**
- L is the number of data bits, L should be divisible by k

\[
R_{\text{eff}} = \frac{L}{n\left[\frac{L}{k} + (K - 1)\right]} < R_c
\]

*Example:* \(\mathbf{m} = [101]\)
- \(n=2, K=3, k=1, L=3\)
- \(R_{\text{eff}} = 3/[2(3+3-1)] = 0.3\)
Encoder Representation (1)

- **Vector representation:**
  - Define \( n \) vectors, each with \( Kk \) elements (one vector for each modulo-2 adder). The \( i \)-th element in each vector, is “1” if the \( i \)-th stage in the shift register is connected to the corresponding modulo-2 adder, and “0” otherwise.
  - **Examples:** \( k=1 \)

\[
\begin{align*}
g_1 &= (111) \\
g_2 &= (101) \\
g_3 &= (111)
\end{align*}
\]

\( U = m \otimes g_i \) interlaced with \( m \otimes g_2 \)

\[ g_1 = (100) \]

\[ g_2 = (101) \]

\[ g_3 = (111) \]
Polynomial representation (1):

- Define \( n \) generator polynomials, one for each modulo-2 adder. Each polynomial is of degree \( Kk-1 \) or less and describes the connection of the shift registers to the corresponding modulo-2 adder.
- *Examples:* \( k=1 \)

\[
g_1(X) = g_0^{(1)} + g_1^{(1)} X + g_2^{(1)} X^2 = 1 + X + X^2
\]
\[
g_2(X) = g_0^{(2)} + g_1^{(2)} X + g_2^{(2)} X^2 = 1 + X^2
\]

The output sequence is found as follows:

\[
U(X) = m(X)g_1(X) \text{ interlaced with } m(X)g_2(X)
\]
\[
= m(X)g_1(X) + Xm(X)g_2(X)
\]
Encoder Representation (3)

- Polynomial representation (2):

*Example: \( m = (1 \ 0 \ 1) \)

\[
\begin{align*}
\mathbf{m}(X)g_1(X) &= (1 + X^2)(1 + X + X^2) = 1 + X + X^3 + X^4 \\
\mathbf{m}(X)g_2(X) &= (1 + X^2)(1 + X^2) = 1 + X^4 \\
\mathbf{m}(X)g_1(X) &= 1 + X + 0.X^2 + X^3 + X^4 \\
\mathbf{m}(X)g_2(X) &= 1 + 0.X + 0.X^2 + 0.X^3 + X^4 \\
\mathbf{U}(X) &= (1,1) + (1,0)X + (0,0)X^2 + (1,0)X^3 + (1,1)X^4 \\
\mathbf{U} &= 11 \ 10 \ 00 \ 10 \ 11
\end{align*}
\]
Tree Diagram (1)

One method to describe a convolutional code

Example: $k=1$

The state of the first $(K-1)k$ stages of the shift register:

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

The structure repeats itself after $K$ stages (3 stages in this example).
Tree Diagram (2)

Example: $k=2$

The state of the first $(K-1)k$ stages of the shift register:
- $a=00$
- $b=01$
- $c=10$
- $d=11$

Input bit: 10 11

Output bits: 111 000

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

$g_1 = (1011)$
$g_2 = (1101)$
$g_3 = (1010)$

The state of the first $(K-1)k$ stages of the shift register:
- $a=00$
- $b=01$
- $c=10$
- $d=11$
A convolutional encoder is a finite-state machine:

- The state is represented by the content of the memory, i.e., the \((K-l)k\) previous bits, namely, the \((K-l)k\) bits contained in the first \((K-l)k\) stages of the shift register. Hence, there are \(2^{(K-l)k}\) states.

  - Example: 4-state encoder

  - The output sequence at each stage is determined by the input bits and the state of the encoder.
State Diagram (2)

- A state diagram is simply a graph of the possible states of the encoder and the possible transitions from one state to another. It can be used to show the relationship between the encoder state, input, and output.

- The stage diagram has $2^{(K-1)k}$ nodes, each node standing for one encoder state.

- Nodes are connected by branches
  - Every node has $2^k$ branches entering it and $2^k$ branches leaving it.
  - The branches are labeled with $c$, where $c$ is the output.
  - When $k=1$
    - The solid branch indicates that the input bit is 0.
    - The dotted branch indicates that the input bit is 1.
Example of State Diagram (1)

The possible transitions:
- \( a^0 \rightarrow a; \quad a^1 \rightarrow c \)
- \( b^0 \rightarrow a; \quad b^1 \rightarrow c \)
- \( c^0 \rightarrow b; \quad c^1 \rightarrow d \)
- \( d^0 \rightarrow b; \quad d^1 \rightarrow d \)

Initial state

Input bit: 101
Output bits: 111 001 100

Input bit: 101
Output bits: 111 001 100
Example of State Diagram (2)

Initial state

Input bit: 10 11
Output bits: 111 000
Distance Properties of Convolutional Codes (1)

- The state diagram can be modified to yield information on code distance properties.
- **How to modify the state diagram:**
  - *Split* the state $a$ (all-zero state) into initial and final states, remove the self loop
  - *Label* each branch by the branch gain $D^i$, where $i$ denotes the Hamming weight of the $n$ encoded bits on that branch
- Each path connecting the initial state and the final state represents a non-zero codeword that diverges from and re-emerges with state $a$ (all-zero state) only once.
Example of Modifying the State Diagram
Distance Properties of Convolutional Codes (2)

- **Transfer function** (which represents the input-output equation in the modified state diagram) indicates the distance properties of the convolutional code by

  \[ T(X) = \sum_d a_d D^d \]

  - \( a_d \) represents the number of paths from the initial state to the final state having a distance \( d \).

- The minimum free distance \( d_{\text{free}} \) denotes
  - The minimum weight of all the paths in the modified state diagram that diverge from and re-emerge with the all-zero state \( a \).
  - The lowest power of the transfer function \( T(X) \)
Example of Transfer Function

\[ T(X) = \frac{X_e}{X_a} = \frac{D^6}{1 - 2D^2} = D^6 + 2D^8 + 4D^{10} + 8D^{12} + \cdots \]

\[ a_d = \begin{cases} 2^{(d-6)/2} & \text{(even } d) \\ 0 & \text{(odd } d) \end{cases} \]

\[
\begin{align*}
X_c &= D^3 X_a + DX_b \\
X_b &= DX_c + DX_d \\
X_d &= D^2 X_c + D^2 X_d \\
X_e &= D^2 X_b
\end{align*}
\]
Trellis Diagram

- Trellis diagram is an extension of state diagram which explicitly shows the passage of time.
  - All the possible states are shown for each instant of time.
  - Time is indicated by a movement to the right.
  - The input data bits and output code bits are represented by a unique path through the trellis.
  - The lines are labeled with $c$, where $c$ is the output.
  - After the second stage, each node in the trellis has $2^k$ incoming paths and $2^k$ outgoing paths.
  - When $k=1$
    - The solid line indicates that the input bit is 0.
    - The dotted line indicates that the input bit is 1.
Example of Trellis Diagram (1)

Input bit: 10100
Output bits:
111 001 100 001 011
Example of Trellis Diagram (2)

Input bit: 10 11 00
Output bits: 111 000 011

\[ K=2, k=2, n=3 \text{ convolutional code} \]
Maximum Likelihood Decoding

- Given the received code word \( r \), determine the most likely path through the trellis. (maximizing \( p(r|c') \))
  - Compare \( r \) with the code bits associated with each path
  - Pick the path whose code bits are “closest” to \( r \)
  - Measure distance using either Hamming distance for hard-decision decoding or Euclidean distance for soft-decision decoding
  - Once the most likely path has been selected, the estimated data bits can be read from the trellis diagram
Example of Maximum Likelihood Decoding

**hard-decision**

Received sequence → 10

ML path with minimum Hamming distance of 2

<table>
<thead>
<tr>
<th>path</th>
<th>code sequence</th>
<th>Hamming distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>00 00 00 00 00</td>
<td>5</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>00 00 11 10 11</td>
<td>6</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>00 11 10 11 00</td>
<td>2</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>00 11 01 01 11</td>
<td>7</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>11 10 11 00 00</td>
<td>6</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>11 10 00 10 11</td>
<td>7</td>
</tr>
<tr>
<td>1 1 0 0 0</td>
<td>11 01 01 11 00</td>
<td>3</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>11 01 10 01 11</td>
<td>4</td>
</tr>
</tbody>
</table>

All path metrics should be computed.
The Viterbi Algorithm (1)

- A breakthrough in communications in the late 60’s
  - Guaranteed to find the ML solution
    - However the complexity is only $O(2^K)$
    - Complexity does not depend on the number of original data bits
  - Is easily implemented in hardware
    - Used in satellites, cell phones, modems, etc
    - Example: Qualcomm Q1900
The Viterbi Algorithm (2)

- Takes advantage of the structure of the trellis:
  - Goes through the trellis one stage at a time
  - At each stage, finds the most likely path leading into each state (*surviving path*) and discards all other paths leading into the state (*non-surviving paths*)
  - Continues until the end of trellis is reached
  - At the end of the trellis, traces the most probable path from right to left and reads the data bits from the trellis

- Note that in principle whole transmitted sequence must be received before decision. However, *in practice storing of stages with length of 5K is quite adequate*
The Viterbi Algorithm (3)

**Implementation:**

1. **Initialization:**
   - Let $M_t(i)$ be the path metric at the $i$-th node, the $t$-th stage in trellis
   - Large metrics corresponding to likely paths; small metrics corresponding to unlikely paths
   - Initialize the trellis, set $t=0$ and $M_0(0)=0$

2. At stage $(t+1)$,
   - **Branch metric calculation**
     - Compute the metric for each branch connecting the states at time $t$ to states at time $(t+1)$
     - The metric is related to the likelihood probability between the received bits and the code bits corresponding to that branch: $p(r_{(t+1)}|c'_{(t+1)})$
The Viterbi Algorithm (4)

Implementation (cont’d):

2. At stage \((t+1)\),
   - **Branch metric calculation**
     - In hard decision, the metric could be the number of same bits between the received bits and the code bits
   - **Path metric calculation**
     - For each branch connecting the states at time \(t\) to states at time \((t+1)\), add the branch metric to the corresponding partial path metric \(M_t(i)\)
   - **Trellis update**
     - At each state, pick the most likely path which has the largest metric and delete the other paths
     - Set \(M_{(t+1)}(i)\) = the largest metric corresponding to the state \(i\)
The Viterbi Algorithm (5)

- **Implementation (cont’d):**

  3. Set $t = t + 1$; go to step 2 until the end of trellis is reached
  4. Trace back
     - Assume that the encoder ended in the all-zero state
     - The most probable path leading into the last all-zero state in the trellis has the largest metric
       - Trace the path from right to left
       - Read the data bits from the trellis
Examples of Hard-Decision Viterbi Decoding (1)

\[ r = c \oplus e \]

Error indicator:
- \( e = 0 \) no error (prob. 1-\( p \))
- \( e = 1 \) error (prob. \( p \))

Binary Symmetric Channel
\[ p = Q\left( \frac{2\varepsilon_b}{N_0} \right) \]
Examples of the Hard-Decision Viterbi Decoding (2)

Non-surviving paths are denoted by dashes lines.

Correct decoding

\[ c = (111 \ 000 \ 001 \ 001 \ 111 \ 001 \ 111 \ 110) \]

\[ r = (1\overline{0}1 \ \overline{1}00 \ 001 \ 0\overline{1}1 \ 111 \ \overline{1}01 \ 111 \ 110) \]

\[ c' = (111 \ 000 \ 001 \ 001 \ 111 \ 001 \ 111 \ 110) \]
Examples of the Hard-Decision Viterbi Decoding (3)

Non-surviving paths are denoted by dashes lines.

Path metrics

Error event

\[
c = (111, 000, 001, 001, 111, 001, 111, 110)
\]

\[
r = (\overline{101}, \overline{100}, 001, 0\overline{11}, 11\overline{0}, \overline{110}, 111, 110)
\]

\[
c' = (111, \overline{11\overline{1}}, 001, \overline{11\overline{1}}, 11\overline{0}, \overline{111}, 111, 110)
\]
Error Rate of Convolutional Codes (1)

- An error event happens when an erroneous path is selected at the decoder

- **Error-event probability:**

\[ P_e \leq \sum_{d=d_{\text{free}}}^{\infty} a_d P_2(d) \]

- \( a_d \rightarrow \) the number of paths with the Hamming distance of \( d \)
- \( P_2(d) \rightarrow \) probability of the path with the Hamming distance of \( d \)

*Depending on the modulation scheme, hard or soft decision*
Error Rate of Convolutional Codes (2)

- BER is obtained by multiplying the error-event probability by the number of data bit errors associated with each error event.

- **BER is upper bounded by:**

\[
P_b \leq \sum_{d=d_{\text{free}}}^{\infty} f(d)a_d P_2(d)
\]

\(f(d)\) → the number of data bit errors corresponding to the erroneous path with the Hamming distance of \(d\)