4F5: Advanced Communications and Coding
Coding Handout 3: Trees, trellises, the Viterbi Algorithm and convolutional codes

Jossy Sayir

Signal Processing and Communications Lab
Department of Engineering
University of Cambridge
jossy.sayir@eng.cam.ac.uk

Michaelmas 2013
(7,4) Hamming Code

Parity-check matrix, \( N = 7, N - K = 3 \):

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

Systematic Parity-check matrix (by Gaussian elimination):

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

Systematic generator matrix:

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

Codewords:

<table>
<thead>
<tr>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 000</td>
</tr>
<tr>
<td>0001 111</td>
</tr>
<tr>
<td>0010 110</td>
</tr>
<tr>
<td>0100 101</td>
</tr>
<tr>
<td>1000 011</td>
</tr>
<tr>
<td>0011 001</td>
</tr>
<tr>
<td>0101 010</td>
</tr>
<tr>
<td>1001 100</td>
</tr>
<tr>
<td>0110 011</td>
</tr>
<tr>
<td>1010 101</td>
</tr>
<tr>
<td>1100 110</td>
</tr>
<tr>
<td>0111 100</td>
</tr>
<tr>
<td>1011 010</td>
</tr>
<tr>
<td>1101 001</td>
</tr>
<tr>
<td>1110 000</td>
</tr>
<tr>
<td>1111 111</td>
</tr>
</tbody>
</table>
Decoding problem

Discrete Memoryless Channel, for example

Hamming codeword transmitted, received word for example

\[ y = (y_1, \ldots, y_7) = (A, D, C, B, E, D, E) \]

What was the encoded information sequence?
Brute Force Decoding

| $x$     | $P_{Y|x}(y|x)$ |
|---------|---------------|
| 0000000 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{16}$ |
| 0001111 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{2}{4}$ |
| 0010110 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{4}$ |
| 0100101 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1000011 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{4}$ |
| 0011001 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 0101010 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{4}$ |
| 1001100 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 0110011 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1010101 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1100110 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 0111100 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1011010 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1101001 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1110000 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |
| 1111111 | $\frac{1}{16} \times \frac{1}{4} \times \frac{1}{16} \times \frac{1}{2}$ |

$y = (y_1, \ldots, y_7) = (A,D,C,B,E,D,E)$
Brute Force Decoding

\[
\begin{array}{|c|c|}
\hline
x & P_{Y|x}(y|x) \\
\hline
0000000 & 2^{-22} \\
0001111 & 2^{-16} \\
0010110 & 2^{-17} \\
0100101 & 2^{-14} \\
1000111 & 2^{-20} \\
0011001 & 2^{-21} \\
0101010 & 2^{-20} \\
1001100 & 2^{-22} \\
0110011 & 2^{-15} \\
1010101 & 2^{-19} \\
1100110 & 2^{-15} \\
0111100 & 2^{-19} \\
1011010 & 2^{-25} \\
1101001 & 2^{-22} \\
1110000 & 2^{-23} \\
1111111 & 2^{-17} \\
\hline
\end{array}
\]

\[
\begin{align*}
\hat{x} &= (0, 1, 0, 0, 1, 0, 1) \\
\text{Information bits: } &0,1,0,0. \\
2^K(N - 1) &\text{ multiplications,} \\
2^K - 1 &\text{ comparisons}
\end{align*}
\]
Practical decoding metric for block ML decoding

- Multiplicative metrics are inconvenient for implementation
- Equivalent practical ML decoding rule, for any $\alpha > 0$:

$$
\hat{x} = \arg\max_{(x_1, \ldots, x_N) \in \mathcal{C}} \log_2 \left( \alpha \prod_{i=1}^{N} P_{Y|X}(y_i|x_i) \right)
$$

$$
= \arg\max_{(x_1, \ldots, x_N) \in \mathcal{C}} \sum_{i=1}^{N} \mu(x_i, y_i), \text{ where } \mu(x_i, y_i) \overset{\text{def}}{=} \log_2(P_{Y|X}(y_i|x_i)) + \beta
$$

where the first step follows because multiplying by a positive constant or taking the logarithm leaves the arg max unchanged, and $\beta = \frac{1}{N} \log_2 \alpha$. $\alpha$ and $\beta$ are picked so that all metrics are non-negative and can be implemented in fixed-point arithmetic.

- Additive decoding metric for our channel:

<table>
<thead>
<tr>
<th>$x$</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Brute Force Decoding with Additive Metric

<table>
<thead>
<tr>
<th>$x$</th>
<th>$\mu(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>$3 \times 0 + 1 + 2 + 0 + 0 + 0 + 0 = 6$</td>
</tr>
<tr>
<td>0001111</td>
<td>$3 \times 0 + 1 + 0 + 3 + 2 + 3 = 12$</td>
</tr>
<tr>
<td>0010110</td>
<td>$3 \times 0 + 1 + 2 + 3 + 2 + 0 = 11$</td>
</tr>
<tr>
<td>0100101</td>
<td>$3 \times 2 + 1 + 2 + 3 + 0 + 3 = 14$</td>
</tr>
<tr>
<td>1000111</td>
<td>$0 \times 0 + 1 + 2 + 0 + 2 + 3 = 8$</td>
</tr>
<tr>
<td>0011001</td>
<td>$3 \times 0 + 1 + 0 + 0 + 0 + 3 = 7$</td>
</tr>
<tr>
<td>0101010</td>
<td>$3 \times 2 + 1 + 0 + 0 + 2 + 0 = 8$</td>
</tr>
<tr>
<td>1001100</td>
<td>$0 \times 2 + 1 + 0 + 3 + 0 + 0 = 6$</td>
</tr>
<tr>
<td>0110011</td>
<td>$3 \times 2 + 1 + 2 + 0 + 2 + 3 = 13$</td>
</tr>
<tr>
<td>1010101</td>
<td>$0 \times 0 + 1 + 2 + 3 + 0 + 3 = 9$</td>
</tr>
<tr>
<td>1100110</td>
<td>$0 \times 2 + 1 + 2 + 3 + 2 + 0 = 10$</td>
</tr>
<tr>
<td>0111100</td>
<td>$3 \times 2 + 1 + 0 + 3 + 0 + 0 = 9$</td>
</tr>
<tr>
<td>1011010</td>
<td>$0 \times 0 + 1 + 0 + 0 + 2 + 0 = 3$</td>
</tr>
<tr>
<td>1101001</td>
<td>$0 \times 2 + 1 + 0 + 0 + 0 + 3 = 6$</td>
</tr>
<tr>
<td>1110000</td>
<td>$0 \times 2 + 1 + 2 + 0 + 0 + 0 = 5$</td>
</tr>
<tr>
<td>1111111</td>
<td>$0 \times 2 + 1 + 0 + 3 + 2 + 3 = 11$</td>
</tr>
</tbody>
</table>

$y = (y_1, \ldots, y_7) = (A, D, C, B, E, D, E)$

$\hat{x} = (0, 1, 0, 0, 1, 0, 1)$

- **Information bits**: 0,1,0,0.
- $2^K(N-1)$ additions,
- $2^K - 1$ comparisons.
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
E_1 & : x_4 + x_5 + x_6 + x_7 = 0 \\
E_2 & : x_2 + x_3 + x_6 + x_7 = 0 \\
E_3 & : x_1 + x_3 + x_5 + x_7 = 0
\end{align*}
\]
Tree and trellis of a Linear Block Code

\[ 0 = x H^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{cases}
    x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\
    x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\
    x_1 + x_3 + x_5 + x_7 = 0 & (E_3)
\end{cases}
\]

\[
\begin{array}{c|c}
    E_1 & \checkmark \\
    E_2 & \checkmark \\
    E_3 & \times \\
\end{array}
\]

\[
\begin{array}{c|c}
    E_1 & \checkmark \\
    E_2 & \checkmark \\
    E_3 & \checkmark \\
\end{array}
\]
Tree and trellis of a Linear Block Code

\[ 0 = x H^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
E_1: x_4 + x_5 + x_6 + x_7 &= 0 \\
E_2: x_2 + x_3 + x_6 + x_7 &= 0 \\
E_3: x_1 + x_3 + x_5 + x_7 &= 0 
\end{align*}
\]
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
E_1 & : x_4 + x_5 + x_6 + x_7 = 0 \\
E_2 & : x_2 + x_3 + x_6 + x_7 = 0 \\
E_3 & : x_1 + x_3 + x_5 + x_7 = 0
\end{align*}
\]
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[ \begin{align*}
x_4 + x_5 + x_6 + x_7 &= 0 \quad (E_1) \\
x_2 + x_3 + x_6 + x_7 &= 0 \quad (E_2) \\
x_1 + x_3 + x_5 + x_7 &= 0 \quad (E_3) \end{align*} \]
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
    x_4 + x_5 + x_6 + x_7 &= 0 \quad (E_1) \\
    x_2 + x_3 + x_6 + x_7 &= 0 \quad (E_2) \\
    x_1 + x_3 + x_5 + x_7 &= 0 \quad (E_3)
\end{align*}
\]

For every codeword \( x \) with prefix \((x_1, x_2, x_3) = (0, 0, 0)\), there is another codeword \( x' \) with prefix \((x'_1, x'_2, x'_3) = (1, 1, 1)\) for which \((x'_4, x'_5, x'_6, x'_7) = (x_4, x_5, x_6, x_7)\)
Tree and trellis of a Linear Block Code

\[ 0 = x H^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \begin{cases} x_4 + x_5 + x_6 + x_7 = 0 \quad (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 \quad (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 \quad (E_3) \end{cases} \]

For every codeword \( x \) with prefix \((x_1, x_2, x_3) = (1, 1, 0)\), there is another codeword \( x' \) with prefix \((x'_1, x'_2, x'_3) = (0, 0, 1)\) for which \((x'_4, x'_5, x'_6, x'_7) = (x_4, x_5, x_6, x_7)\)

etc.

For every codeword \( x \) with prefix \((x_1, x_2, x_3) = (0, 0, 0)\), there is another codeword \( x' \) with prefix \((x'_1, x'_2, x'_3) = (1, 1, 1)\) for which \((x'_4, x'_5, x'_6, x'_7) = (x_4, x_5, x_6, x_7)\)
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[ \begin{cases} x_4 + x_5 + x_6 + x_7 = 0 \quad (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 \quad (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 \quad (E_3) \end{cases} \]

Simplified representation

\[ \begin{array}{ccc}
E_1 & E_2 & E_3 \\
1 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array} \]

(0)

(1)
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
E_1: & \quad x_4 + x_5 + x_6 + x_7 = 0 \\
E_2: & \quad x_2 + x_3 + x_6 + x_7 = 0 \\
E_3: & \quad x_1 + x_3 + x_5 + x_7 = 0
\end{align*}
\]

Simplified representation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

"ROOT" symbol

"TOOR" symbol
Tree and trellis of a Linear Block Code

\[ 0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \]

\[
\begin{align*}
E_1: & \quad x_4 + x_5 + x_6 + x_7 = 0 \\
E_2: & \quad x_2 + x_3 + x_6 + x_7 = 0 \\
E_3: & \quad x_1 + x_3 + x_5 + x_7 = 0
\end{align*}
\]

Simplified representation

\[ \begin{array}{ccc}
E_1 & E_2 & E_3 \\
1 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array} \]

“ROOT” symbol

“TOOR” symbol

© Jossy Sayir (CUED)
Viterbi’s principle

For every codeword \( \mathbf{x} \) with prefix 
\((x_1, x_2, x_3) = (0, 0, 0)\), there is another codeword \( \mathbf{x}' \) with prefix 
\((x'_1, x'_2, x'_3) = (1, 1, 1)\) for which 
\((x'_4, x'_5, x'_6, x'_7) = (x_4, x_5, x_6, x_7)\)

If prefix \((1, 1, 1)\) beats prefix 
\((0, 0, 0)\) in terms of additive metric, 
then the metrics of all codewords 
starting with \((1, 1, 1)\) will be larger 
than the metrics of all codewords 
starting with \((0, 0, 0)\)

When paths merge in a trellis, 
eliminate the prefix with the lower metric!
The Viterbi Algorithm

Let \( M_\ell(s) \) be the metric corresponding to state \( s \) after stage \( \ell \) in the trellis, and 
\( m_\ell(x(s', s)) \) be the additive metric term added between stage \( \ell - 1 \) and \( \ell \) in the trellis when transitioning between state \( s' \) and \( s \).

**Viterbi Decoding**

1. Initialisation \( M_0(0) = 0 \)
2. Add Compare Select (ACS) recursion: for \( \ell = 1, \ldots, L \) and for all states \( s = (s_1, \ldots, s_\nu) \), calculate 
   \[
   M_\ell(s) = \max_{s' \in \Pi(s)} \{ M_{\ell-1}(s') + m_\ell(x(s', s)) \}
   \]
   where \( \Pi(s) \) is the set of parent states to state \( s \) (states \( s' \) that have a connection with \( s \)). Ties in the maximum are resolved by picking a winner at random.
3. Trace back. The maximum metric is \( M_L(0) \), output the input sequence corresponding to \( M_L(0) \).
The Viterbi Decoder in Action

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$\mathbf{y} = (A, D, C, B, E, D, E)$

$y = (\mathbf{A}, \mathbf{D}, \mathbf{C}, \mathbf{B}, \mathbf{E}, \mathbf{D}, \mathbf{E})$
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

### Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

$y = (A, D, C, B, E, D, E)$
The Viterbi Decoder in Action

Metric Table

\[
\begin{array}{c|ccccc}
  y & A & B & C & D & E \\
\hline
  x & 0 & 3 & 2 & 1 & 0 & 0 \\
  & 1 & 0 & 0 & 1 & 2 & 3 \\
\end{array}
\]

\[
y = (A, D, C, B, E, D, E)
\]

© Jossy Sayir (CUED)  Advanced Communications and Coding  Michaelmas 2013
The Viterbi Decoder in Action

<table>
<thead>
<tr>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

$y = (A, D, C, B, E, D, E)$
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>$y$</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$y = (A, D, C, B, E, D, E)$

$y = (A, D, C, B, E, D, E)$
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

\( y = (A, D, C, B, E, D, E) \)
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th>x</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
The Viterbi Decoder in Action

Metric Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ y = (A, D, C, B, E, D, E) \]
Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder.
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all $2^{N-K}$ states in use at each trellis stage, i.e., $(N - 1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length $N$ who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N - 1)$ additions, one comparison.
- Best case: single parity-check code of length $N$, i.e., $H = \begin{bmatrix} 1 & 1 & \ldots & 1 \end{bmatrix}$, has $2^{N-1}$ codewords and hence the brute force decoder requires $(N - 1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N - 1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the brute force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performance.
- Can we design a code with a predictable, e.g., regular, trellis structure?
Definitions

- Linear convolutional codes are defined as a finite state machine (FSM).
- Output $c_{i,j} = \bigoplus_{\ell=0}^{\nu} g_{i,\ell} b_{i-\ell}$, $i = 1, \ldots, N$ is the output generator index and $j$ denotes the time index, where the symbol $\bigoplus$ denotes sum is in the binary field.
- Common representations are: state diagrams and trellis.
Linear Convolutional Codes
Shift-Register Representation

Example (4 states rate $R = \frac{1}{2}$)

- Rate $R = \frac{K}{N} = \frac{1}{2}$, number of states $2^\nu = 4$
- State: content of the shift register $s = (s_1, s_2)$
- Generators (in octal form) $(5, 7)_8 = (101, 111)$

Note that the state is not linked to the parity-check matrix as it was for linear block codes (there are many ways to define the state of a code)

Note also that this encoder generates two code digits per state transition, as opposed to our previous trellis diagram for general linear block codes that generated only one code digit per state transition
Notation: $b_i/c_{2i}c_{2i+1}$ where $b_i$ is the encoded input bit corresponding to a state transition, and $c_{2i}, c_{2i+1}$ are the resulting code digits for the state transition.
Trellis of a Convolutional Code

- Trellis consists of identical copies of the trellis module (except near root and toor)
- Trellis termination drives the FSM back to its all-zero state and thus encodes no information. For feedforward (non-recursive) convolutional encoders, this is achieved simply by padding zeros at the end of the information sequence.
- Note that states can always be ordered so the outgoing edge corresponding to a “1” is above the edge corresponding to a “0”, so there is no need for colour coding.
Viterbi Algorithm for Convolutional Codes

Example
- Transmission over the channel defined previously
- Received sequence
  \text{ABACDEDEABADEDCADC}

<table>
<thead>
<tr>
<th>Additive Metric Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>( x )</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

\( m \) path metric
\( M \) node metric
\( \bar{M} \) tie at node

\( y = (AB, AC, DE, DE, AB, AD, ED, CA, DC) \)
Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

<table>
<thead>
<tr>
<th>Additive Metric Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

$m$ path metric

$M$ node metric

$M$ tie at node

$\overline{y} = (AB, AC, DE, DE, AB, AD, ED, CA, DC)$
Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$m$ path metric
$M$ node metric
$M$ tie at node

$y = \{AB, AC, DE, DE, AB, AD, ED, CA, DC\}$
Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence \( \text{ABACDEDEABADEDCADC} \)

<table>
<thead>
<tr>
<th>Additive Metric Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

\( m \) path metric
\( M \) node metric
\( \_M \) tie at node

\( \mathbf{y} = (\text{AB, AC, DE, DE, AB, AD, ED, CA, DC}) \)
Viterbi Algorithm for Convolutional Codes

**Example**

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

**Additive Metric Table**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$m$ path metric
$M$ node metric
$M$ tie at node

$\bar{y} = (AB, AC, DE, DE, AB, AD, ED, CA, DC)$
Example

- Transmission over the channel defined previously
- Received sequence \( ABACDEDEABADEDCADC \)

Additive Metric Table

<table>
<thead>
<tr>
<th>( \chi )</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\( m \) path metric
\( M \) node metric
\( \bar{M} \) tie at node

\( \bar{y} = (AB, AC, DE, DE, AB, AD, ED, CA, DC) \)
Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$m$ path metric
$M$ node metric
$M$ tie at node

$y \rightarrow (AB, AC, DE, DE, AB, AD, ED, CA, DC)$
Example

- Transmission over the channel defined previously
- Received sequence: ABACDEDEABADEDCADC

Additive Metric Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[
\hat{y} = (\text{AB, AC, DE, DE, AB, AD, ED, CA, DC})
\]
\[
\hat{u} = (0, 0, 1, 0, 1, 0, 0, 0, 0)
\]

No ties on winning path: ML solution is unique
Performance of Trellis Decoding and Convolutional Codes

- We have seen: linear block codes can be decoded using a trellis but the complexity of this trellis is in general prohibitive as the number of states increases with the block length.
- Convolutional codes have a regular trellis module and hence their trellis complexity remains constant per time instant.
- One can show that the performance of convolutional codes is bounded for a given trellis module and does not improve as the length of the trellis goes to infinity.
- Convolutional codes by themselves are not capacity-approaching for a given trellis module / FSM, and their performance is essentially determined by the number of states of the associated FSM.