6.02 Spring 2009
Lecture #10

- state machines & trellises
- path and branch metrics
- Viterbi algorithm: add-compare-select
- hard decisions vs. soft decisions

**Example**

- Using k=3, rate ½ convolutional code
- Received: 11101100011000
- Some errors have occurred...
- What's the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

<table>
<thead>
<tr>
<th>Msg</th>
<th>Xmit</th>
<th>Rcvd</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00000000000000</td>
<td>11101100011000</td>
<td>7</td>
</tr>
<tr>
<td>0001</td>
<td>00000001111100</td>
<td>0001</td>
<td>8</td>
</tr>
<tr>
<td>0010</td>
<td>00001111110000</td>
<td>0010</td>
<td>8</td>
</tr>
<tr>
<td>0011</td>
<td>00001101011100</td>
<td>0011</td>
<td>6</td>
</tr>
<tr>
<td>0100</td>
<td>00111101110000</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0101</td>
<td>01111101111000</td>
<td>0110</td>
<td>7</td>
</tr>
<tr>
<td>0110</td>
<td>01101001011100</td>
<td>0111</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>01100100111100</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>1000</td>
<td>11110000000000</td>
<td>1001</td>
<td>5</td>
</tr>
<tr>
<td>1001</td>
<td>11110111111100</td>
<td>1010</td>
<td>7</td>
</tr>
<tr>
<td>1010</td>
<td>11110111111100</td>
<td>1011</td>
<td>2</td>
</tr>
<tr>
<td>1011</td>
<td>11110100011100</td>
<td>1100</td>
<td>5</td>
</tr>
<tr>
<td>1100</td>
<td>11100110011100</td>
<td>1101</td>
<td>6</td>
</tr>
<tr>
<td>1101</td>
<td>11100100111100</td>
<td>1110</td>
<td>3</td>
</tr>
<tr>
<td>1110</td>
<td>11101001011100</td>
<td>1111</td>
<td>6</td>
</tr>
<tr>
<td>1111</td>
<td>11101010011100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most likely: 1011

**Finding the Most-likely Path**

- Path metric: number of errors on most-likely path to given state (min of all paths leading to state)
- Branch metric: for each arrow, the Hamming distance between received parity and expected parity

---

**State Machines & Trellises**

- Example: k=3, rate ½ convolutional code
  - G₀ = 111: p₀ = 1* x[n] ⊕ 1* x[n-1] ⊕ 1* x[n-2]
  - G₁ = 110: p₁ = 1* x[n] ⊕ 1* x[n-1] ⊕ 0* x[n-2]
- States labeled with x[n-1] x[n-2]
- Arcs labeled with x[n] / p₀ p₁
Viterbi Algorithm

• Compute branch metrics for next set of parity bits
• Compute path metric for next column
  – add branch metric to path metric for old state
  – compare sums for paths arriving at new state
  – select path with smallest value (fewest errors, most likely)

Example (cont’d.)

• After receiving 3 pairs of parity bits we can see that all ending states are equally likely
• Power of convolutional code: use future information to constrain choices about most likely events in the past

Survivor Paths

• Notice that some paths don’t continue past a certain state
  – Will not participate in finding most-likely path: eliminate
  – Remaining paths are called survivor paths
  – When there’s only one path: we’ve got a message bit!

Example (cont’d.)

• When there are “ties” (sum of metrics are the same)
  – Make an arbitrary choice about incoming path
  – If state is not on most-likely path: choice doesn’t matter
  – If state is on most-likely path: choice may matter and error correction has failed (mark state with underline to tell)
Example (cont’d.)

- When we reach end of received parity bits
  - Each state’s path metric indicates how many errors have happened on most-likely path to state
  - Most-likely final state has smallest path metric
  - Ties means end of message uncertain (but survivor paths may merge to a unique path earlier in message)

Traceback

- Use most-likely path to determine message bits
  - Trace back through path: message in reverse order
  - Message bit determined by high-order bit of each state (remember that came from message bit when encoding)
  - Message in example: 1011000 (w/ 2 transmission errors)

Hard Decisions

- As we receive each bit it’s immediately digitized to “0” or “1” by comparing it against a threshold voltage
  - We lose the information about how “good” the bit is: a “1” at .9999V is treated the same as a “1” at .5001V
- The branch metric used in the Viterbi decoder is the Hamming distance between the digitized received voltages and the expected parity bits
  - This is called hard-decision Viterbi decoding
- Throwing away information is (almost) never a good idea when making decisions
  - Can we come up with a better branch metric that uses more information about the received voltages?

Soft Decisions

- Let’s limit the received voltage range to [0.0,1.0]
  - \( V_{\text{eff}} = \max(0.0, \min(1.0, V_{\text{received}})) \)
  - Voltages outside this range are “good” 0’s or 1’s
- Define our “soft” branch metric as the Euclidian distance between received \( V_{\text{eff}} \) and expected voltages
  - Soft-decision decoder chooses path that minimizes sum of Euclidian distances between received and expected voltages
  - Different branch metric but otherwise the same recipe
Channel Coding Summary

- Add redundant info to allow error detection/correction
  - CRC to detect error-transmission (our safety net for catching undiscovered or uncorrectable errors)
  - Block codes: multiple parity bits, RC codes: oversampled polynomials
  - Convolutional codes: continuous streams of parity bits

- Error correction
  - Block codes: use parity errors to triangulate which bits are in error
  - RS codes: use subsets of bits to vote for message, majority rules!
  - Convolutional codes: use Viterbi algorithm to find most-likely message