### Using Convolutional Codes

- **Transmitter**
  - Begins transmission at its starting state, processes message bit-by-bit:
  - For each message bit, it makes a state transition and sends parity bits.

- **Receiver**
  - Doesn’t have direct knowledge of the transmitter’s state transitions; only knows (possibly corrupted) received parity bits.
  - Must find the most likely sequence of transmitter states that could have generated the received parity bits, $p_i$.

If BER is $< 1/2$, then:
- Most likely message sequence is the one that generated the sequence of parity bits with the smallest Hamming distance from the actual received $p_i$.

### Example

- Using K=3, rate ½ convolutional code:
- States labeled with $x[n-1] x[n-2]$
- Arcs labeled with $x[n] / p_0 p_1$
- msg=101100, xmit = 11 11 01 00 01 10

### Viterbi Algorithm

- Want: Most likely message sequence
- Have: (possibly corrupted) received parity sequences
- Viterbi algorithm for a given K and r:
  - Works incrementally to compute most likely message sequence
  - Uses two metrics:
- Branch metric: BM($xmit, rcvd$) proportional to likelihood that the transmitter sent $xmit$ given that we’ve received $rcvd$.
  - “Hard decision”: use digitized bits, compute Hamming distance between $xmit$ and $rcvd$. Smaller distance is more likely if BER < 1/2.
  - “Soft decision”: use function of received voltages directly
- Path metric: PM[$s, i$] for each state $s$ of the $2^K - 1$ transmitter states and bit time $i$ where $0 \leq i < \text{len(message)}$
  - PM[$s, i$] = most likely sum of BM($xmit, rcvd$) over all message sequences that place transmitter in state $s$ at time $i$
  - PM[$s, i+1$] computed from PM[$s, i$] and $p_0[i], …, p_{r-1}[i]$
6.02 Fall 2010

2) \(\min(3 + 2, 2 + 1) = 3\)

Example:

\(\text{pm}[11,i+1] + 1\) = \(\min(\text{pm}[10,i] + 2, \text{pm}[01,i+1])\)

Q: What's the most likely state \(s\) for the transmitter at time \(i\)?
A: state 00 (smallest \(\text{pm}[s,i]\))

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

Computing \(\text{pm}[s,i+1]\) cont'd.

Example cont'd: to arrive in state 01 at time \(i+1\), either
1) The transmitter was in state 10 at time \(i\) and the \(i\)th message bit was a 0. If that's the case, the transmitter sent 11 as the parity bits and there were 2 bit errors since we received 00. Total bit errors = \(\text{pm}[10,i] + 2 = 5\) OR
2) The transmitter was in state 11 at time \(i\) and the \(i\)th message bit was a 0. If that's the case, the transmitter sent 01 as the parity bits and there was 1 bit error since we received 00. Total bit errors = \(\text{pm}[11,i] + 1 = 3\)
Which is more likely?

Computing \(\text{pm}[s,i+1]\) cont’d.

Formalizing the computation:

\[\text{pm}[s,i+1] = \min(\text{pm}[\alpha,i] + \text{bm}[\alpha\rightarrow s], \text{pm}[\beta,i] + \text{bm}[\beta\rightarrow s])\]

Example:

\(\text{pm}[01,i+1] = \min(\text{pm}[10,i] + 2, \text{pm}[11,i] + 1)\)

\(= \min(3, 2+1) = 3\)

Notes:
1) Remember which arc was min; saved arcs will form a path through trellis
2) If both arcs have same sum, break tie arbitrarily (e.g., when computing \(\text{pm}[11,i+1]\))
• 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)

When we reach end of received parity bits

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!

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 mean end of message uncertain (but survivor paths may merge to a unique path earlier in message)

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 underscore to self

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

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: 101100 (w/ 2 transmission errors)
**Viterbi Algorithm with Hard Decisions**

- Branch metrics measure the likelihood by comparing received parity bits to possible transmitted parity bits computed from possible messages.
- Path metric $PM[s,i]$ proportional to likelihood of transmitter being in state $s$ at time $i$, assuming the mostly likely message of length $i$ that leaves the transmitter in state $s$.
- Most likely message? The one that produces the most likely $PM[s,N]$.
- At any given time there are $2^{K-1}$ most-likely messages we’re tracking → time complexity of algorithm grows exponentially with constraint length $K$.

---

**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 9.999V 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 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 Decision Decoding**

- In practice, the receiver gets a voltage level, $V$, for each received parity bit
  - Sender sends 0 or 1 Volt; $V$ in $[0,\infty)$ assuming additive Gaussian noise
- Idea: Pass received voltages to decoder **before** digitizing
- Define a “soft” branch metric as the square of the Euclidian distance between received voltages and expected voltages
- Soft-decision decoder chooses path that minimizes sum of the squares of the Euclidian distances between received and expected voltages
  - Different BM & PM values, but otherwise the same algorithm