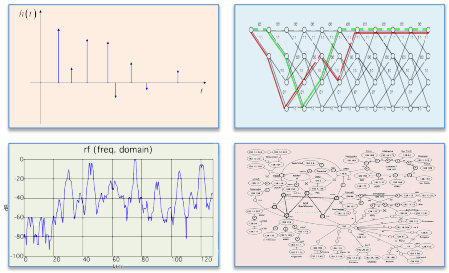


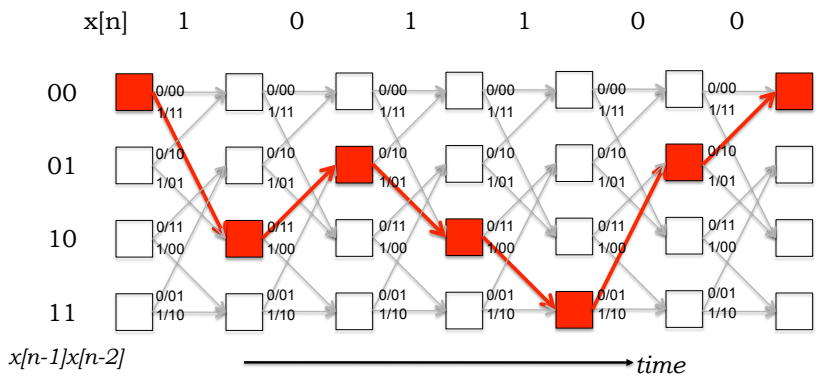
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #11

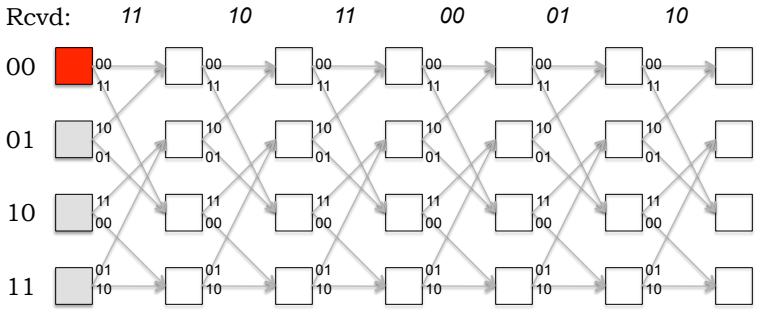
- state machines & trellises
- path and branch metrics
- Viterbi algorithm: add-compare-select
- hard decisions vs. soft decisions

Trellis View @ Transmitter



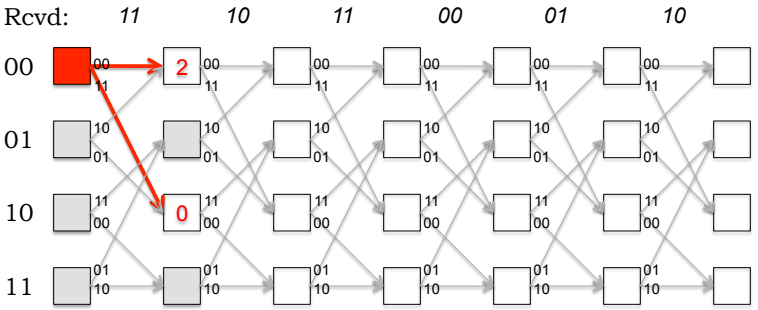
Send: 11 11 01 00 01 10

Finding the Most-likely Path



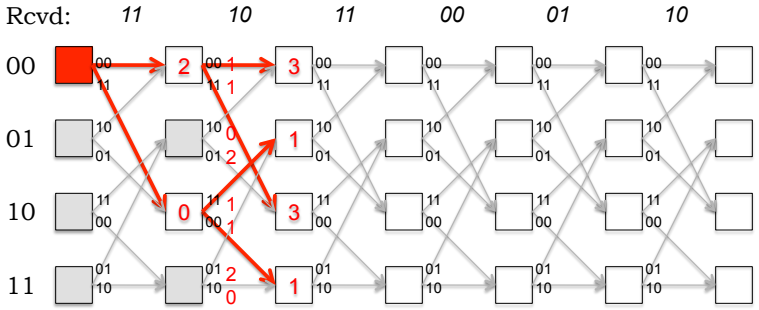
Given the received parity bits, the receiver must find the most-likely sequence of transmitter states, i.e., the path through the trellis that minimizes the Hamming distance between the received parity bits and the parity bits the transmitter would have sent had it followed that state sequence.

Processing 1st pair of parity bits



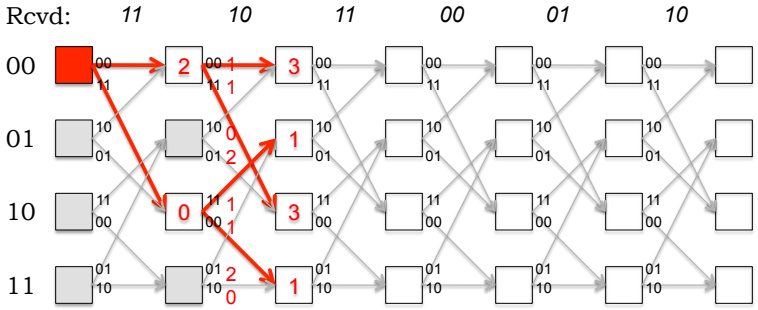
- Transmitter is known to be in state 00 at start of message.
- If next message bit 0, next state is 00, transmit 00
  - So receiving 11 means there have been two errors in the channel
- If next message bit is 1, next state is 10, transmit 11
  - So receiving 11 means there have been no errors in the channel

Processing 2nd pair of parity bits



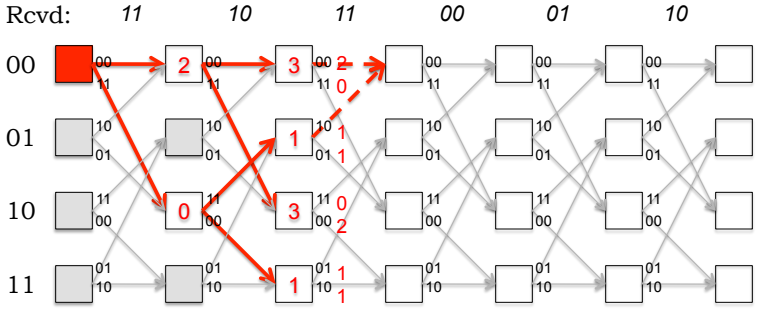
- Consider transitions for each possible transmitter state
- Compute Hamming distance between what would have been transmitted and what was actually received → indicates number of errors that had to have happened in this case.
- Enter total errors along path in each destination state in next column of the trellis, color transition arc red

Branch Metrics, Path Metrics



- The red numbers for HD(xmit,rcvd) are called *branch metrics* → indicate number of errors if this arc was the true path
- The numbers in the trellis boxes are called *path metrics* → indicate total number of errors along path ending at box
- The red arrows indicate sequence of transmitter states that end at a particular column and state.

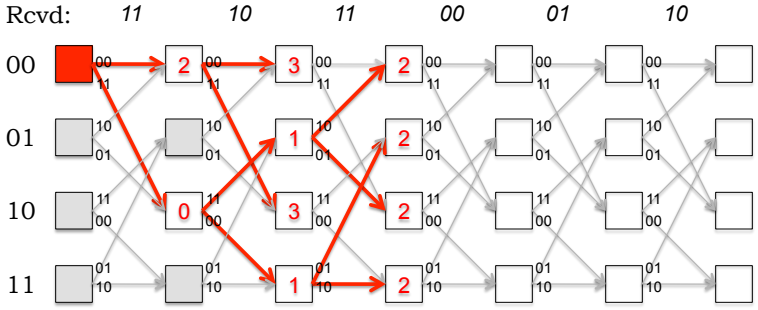
Processing 3rd pair of parity bits



What if there are two possible paths to a particular state?

- Consider each path separately: compute total errors along each path (e.g., one path to 00 has 5 total errors, the other 2 errors)
- Select path with fewest total errors as the *most-likely path*
- Steps: add branch metrics, compare total errors, select path

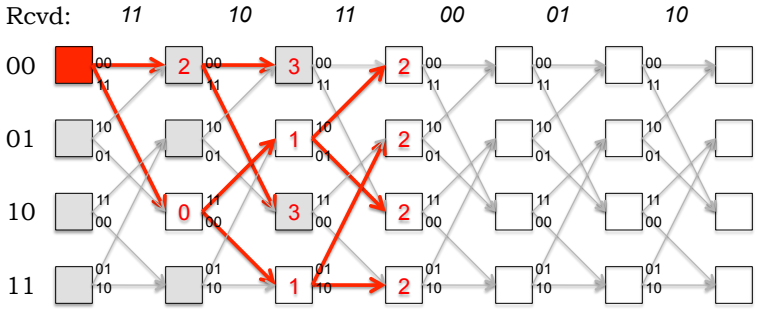
What Can We Tell About Message?



Hmm, at this point all ending states are equally likely, each corresponding to a path with 2 errors. Receiver doesn't (yet) have enough information to decode first 3 message bits.

Can receiver tell anything about the message?

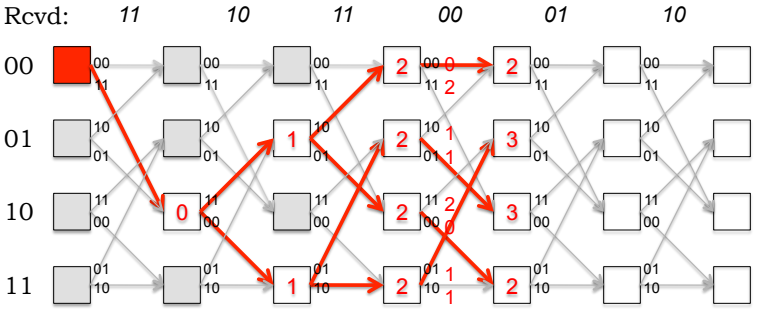
Survivor Paths



Receiver can make some deductions:

Some earlier states are no longer part of *any* most-likely path. We can eliminate partial paths leading to those states since they will not be part of the final most-likely path. Do this recursively…

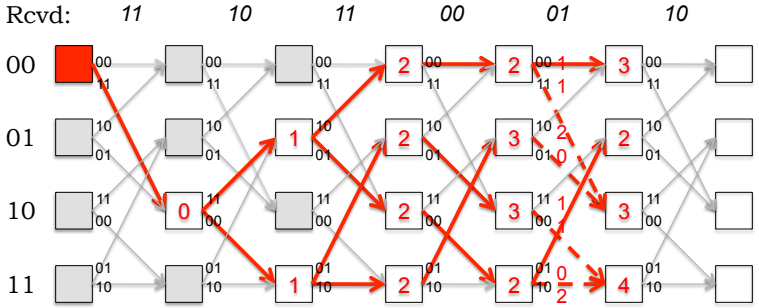
Processing 4th pair of parity bits



The usual:

- Add branch metrics to previous total errors
- Compare paths arriving at destination state
- Select path with smallest total errors as most-likely path

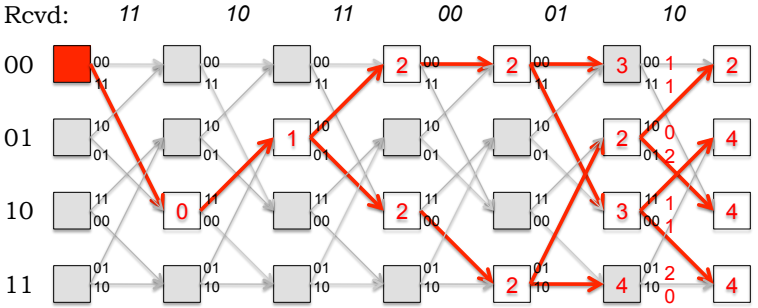
Processing 5th pair of parity bits



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

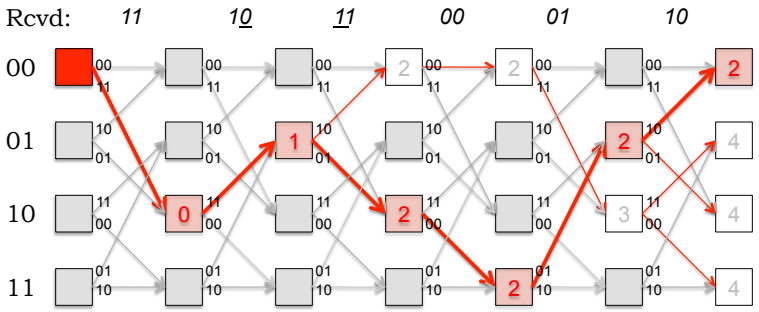
Processing 6th pair of parity bits



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)

Traceback



Use most-likely path to determine message bits, tracing path backwards starting with most-likely end state. In this example, the transmitter states along most-likely path, from left to right:

10 01 10 11 01 00

Message bits from high-order state bit:

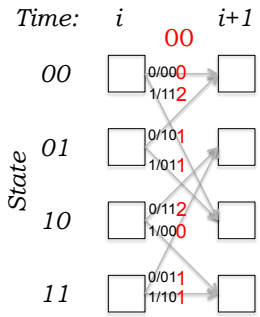
101100

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) measures likelihood that 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 is small
  - "Soft decision": use received voltages (more later…)
- Path metric: PM[s,i] for each state *s* of the 2^{k-1} transmitter states and bit time *i* where $0 \le i < \text{len(message)}$.
  - PM[s,i] = most likely BM(xmit_m,received parity) over all message sequences *m* that leave transmitter in state *s* at time *i*
  - PM[s,i+1] computed from PM[s,i] and $p_0[i],\ldots,p_{r-1}[i]$

Hard-decision Branch Metric

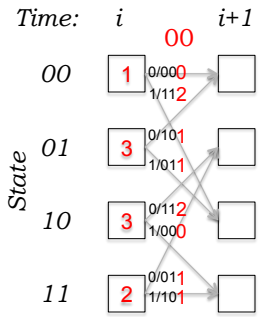
- BM = Hamming distance between expected parity bits and received parity bits
- Compute BM for each transition arc in trellis
  - Example: received parity = 00
  - BM(00,00) = 0
    BM(01,00) = 1
    BM(10,00) = 1
    BM(11,00) = 2
- Will be used in computing PM[s,i+1] from PM[s,i].
- We'll want most likely BM, which, since we're using Hamming distance, means minimum BM.



Computing PM[s,i+1]

Starting point: we've computed PM[s,i], shown graphically as label in trellis box for each state at time *i*.

Example: PM[00,i] = 1 means there was 1 bit error detected when comparing received parity bits to what would have been transmitted when sending the most likely message, considering all messages that leave the transmitter in state 00 at time i.



Q: What's the most likely state *s* for the transmitter at time *i*?

A: state 00 (smallest PM[s,i])

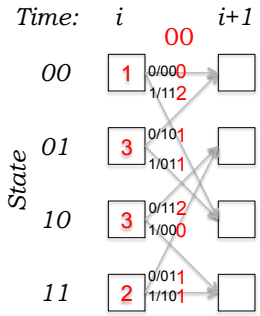
Computing PM[s,i+1] cont'd.

Q: If the transmitter is in state s at time i+1, what state(s) could it have been in at time i?

A: For each state s, there are two predecessor states α and β in the trellis diagram

Example: for state 01, α=10 and β=11.

Any message sequence that leaves the transmitter in state s at time i+1 must have left the transmitter in state α or state β at time i.

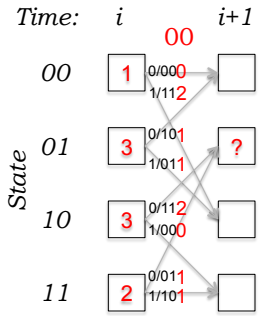


Computing PM[s,i+1] cont'd.

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 = 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 = PM[11,i] + 1 = 3

Which is mostly likely?



Computing PM[s,i+1] cont'd.

Formalizing the computation:

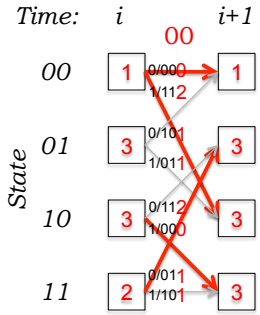
PM[s,i+1] = min(PM[α,i] + BM[α→s],
PM[β,i] + BM[β→s])

Example:

PM[01,i+1] = min(PM[10,i] + 2,
PM[11,i] + 1)
= min(3+ 2,2+1) = 3

Notes:

1) Remember with 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 PM[11,i+1])



Viterbi Algorithm Summary

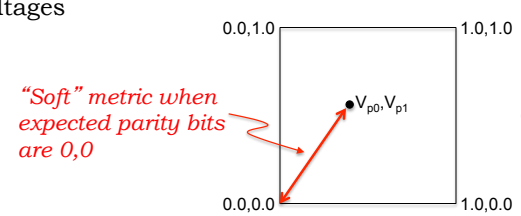
- Branch metrics measure the likelihood by comparing received parity bits to possible transmitted parity bits computed from possible messages.
- Path metric PM[s,i] measures the likelihood of the transmitter being in state s at time i assuming the mostly likely message of length i that leaves the transmitter 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 .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?

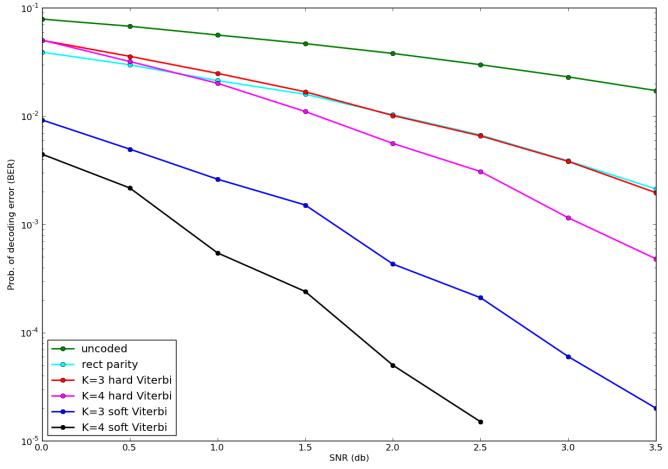
Soft Decisions

- Let's limit the received voltage range to [0.0,1.0]
  - $V_{eff} = \max(0.0, \min(1.0, V_{received}))$
  - Voltages outside this range are "good" 0's or 1's
- Define our "soft" branch metric as the square of the Euclidian distance between received V_{eff} and expected voltages

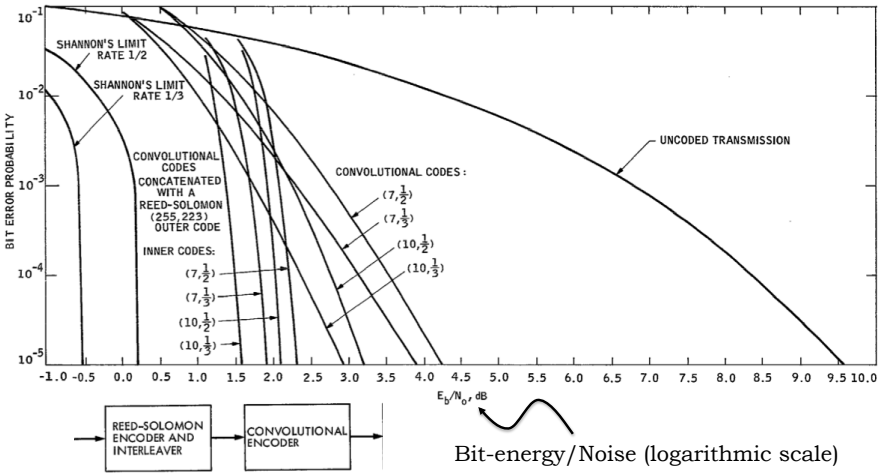


- Soft-decision decoder chooses path that minimizes sum of the squares of the Euclidian distances between received and expected voltages
  - Different branch metric but otherwise the same recipe

More Work, Better BER



Code performance



Source: Butman, Deutsch, Miller, "Performance of Concatenated Codes for Deep Space Missions"