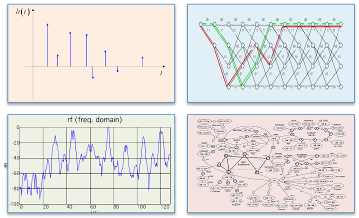
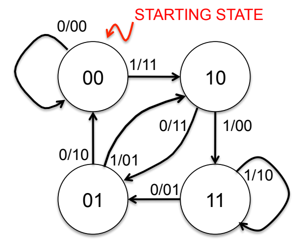


INTRODUCTION TO EECS II
DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2011
Lecture #8

- State machines & trellises (recap)
- Path and branch metrics
- Viterbi decoding of convolutional codes
- Hard decision vs. soft decision decoding
- Puncturing, free distance, and performance

State Machine View



The state machine is the same for all K=3 codes. Only the p_i labels change depending on number and values for the generator polynomials.

- Example: K=3, rate ½ convolutional code
- States labeled with x[n-1] x[n-2]
- Arcs labeled with $x[n]/p_0p_1$
- msg=101100; xmit = 11 11 01 00 01 10

Using Convolutional Codes

- Transmitter
  - Beginning at starting state, processes message bit-by-bit
  - For each message bit: makes a state transition, sends parity bits
- Receiver
  - Doesn't have direct knowledge of transmitter's state transitions; only knows (possibly corrupted) received parity bits
  - Must find 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

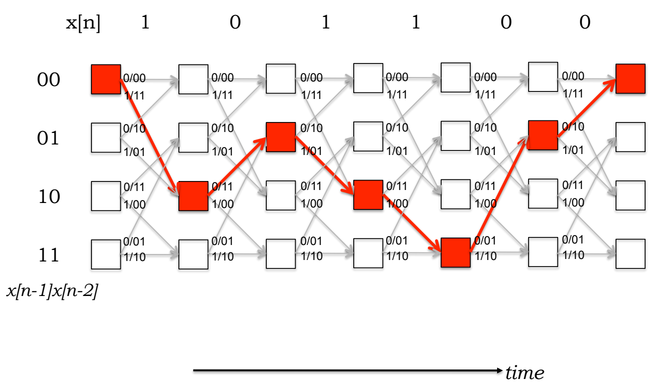
- Using K=3, rate ½ code from earlier slides
- Received: 111011000110
- Some errors have occurred…
- What's the 4-bit message?
- Look for message whose xmit bits are closest to rcvd bits

Msg	Xmit*	Rcvd	d
0000	000000000000		7
0001	000000111110		8
0010	000011111000		8
0011	000011010110		4
0100	001111100000		6
0101	001111011110		5
0110	001101001000		7
0111	001100100110	111011000110	6
1000	111110000000		4
1001	111110111110		5
1010	111101111000		7
1011	111101000110		2
1100	110001100000		5
1101	110001011110		4
1110	110010011000		6
1111	110010100110		3

*Msg padded with 2 zeros before transmission

Most likely: 1011

Trellis View @ Transmitter

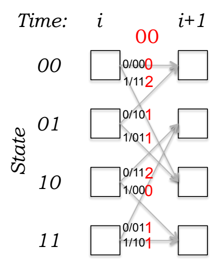


Viterbi Algorithm

- 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 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 $0 \le i <$ len(message).
  - PM[s,i] = most likely sum of BM($xmit_m$,received parity) over all message sequences *m* that place 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 want to use the most likely BM, which means minimum BM.



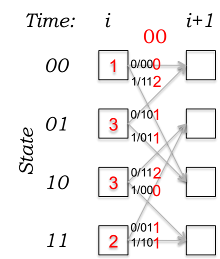
Computing PM[s,i+1]

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 place 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])



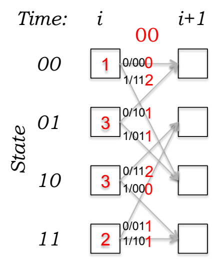
Computing PM[s,i+1] cont'd.

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.

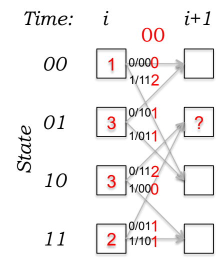
Example cont'd: to arrive in state 01 at time i+1, either

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 more likely?



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

Formalizing the computation:

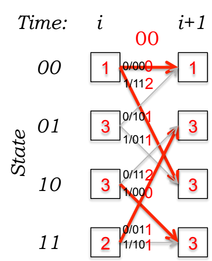
PM[s,i+1] = min(PM[α,i] + BM[α→s],
PM[β,i] + BM[β→s])

Example:

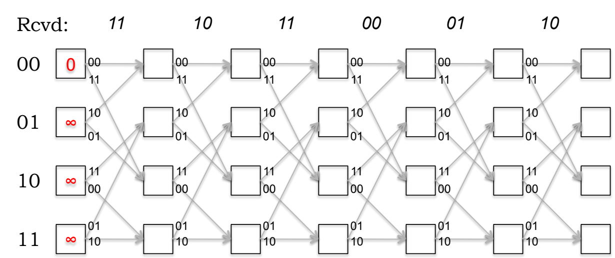
PM[01,i+1] = min(PM[10,i] + 2,
PM[11,i] + 1)
= min(3+ 2,2+1) = 3

Notes:

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

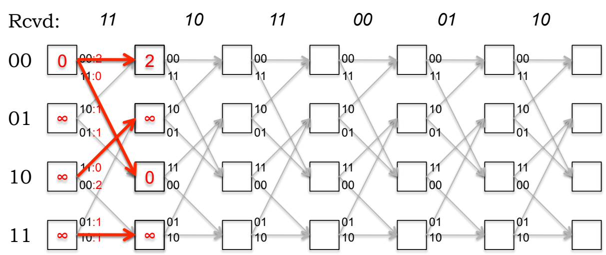


Finding the Maximum-Likelihood Path



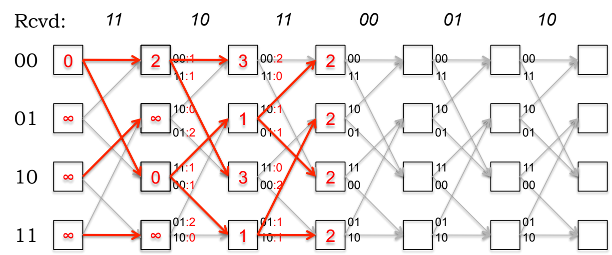
- Path metric: number of errors on maximum-likelihood path to given state (min of all paths leading to state)
- Branch metric: for each arrow, the Hamming distance between received parity and expected parity

Viterbi Algorithm



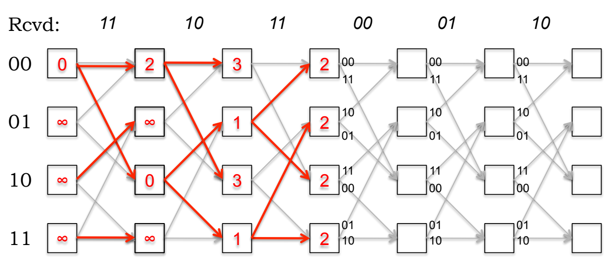
- Compute branch metrics for next set of parity bits
- Compute path metric for next column
  - *add* branch metric to path metric for old state
  - *compare* sums for paths arriving at new state
  - *select* path with smallest value (fewest errors, most likely)

Example (cont'd.)



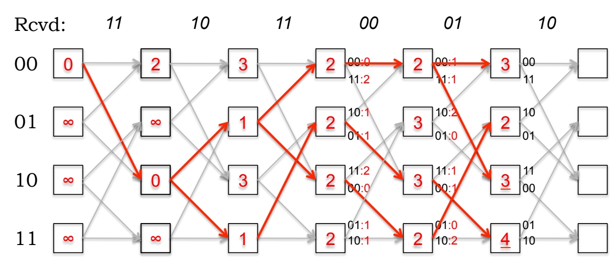
- After receiving 3 pairs of parity bits we can see that all ending states are equally likely
- Power of convolutional code: use future information to constrain choices about most likely events in the past

Survivor Paths



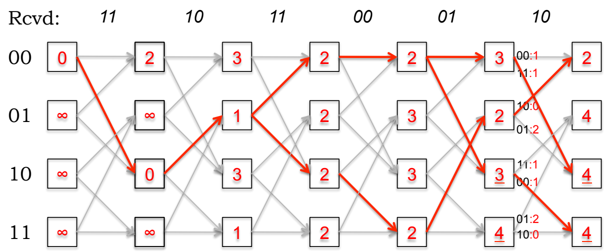
- Notice that some paths don't continue past a certain state
  - Will not participate in finding most-likely path: eliminate
  - Remaining paths are called *survivor paths*
  - When there's only one path: we've got a message bit!

Example (cont'd.)



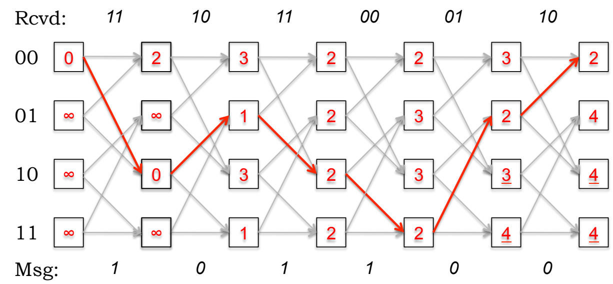
- When there are "ties" (sum of metrics are the same)
  - Make an arbitrary choice about incoming path
  - If state is not on most-likely path: choice doesn't matter
  - If state is on most-likely path: choice may matter and error correction has failed *(mark state with underline to tell)*

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: 101100 (w/ 2 transmission errors)

Viterbi Algorithm with Hard Decisions

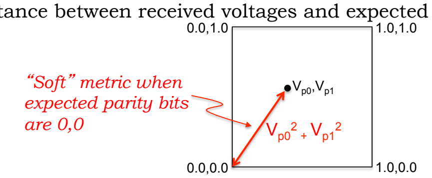
- Branch metrics measure the likelihood by comparing received parity bits to possible transmitted parity bits computed from possible messages.
- Path metric PM[s,i] proportional to likelihood of transmitter being in state s at time *i*, assuming the mostly likely message of length i that leaves the transmitter 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 .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 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 V0 or V1 volts; V in (-∞,∞) 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

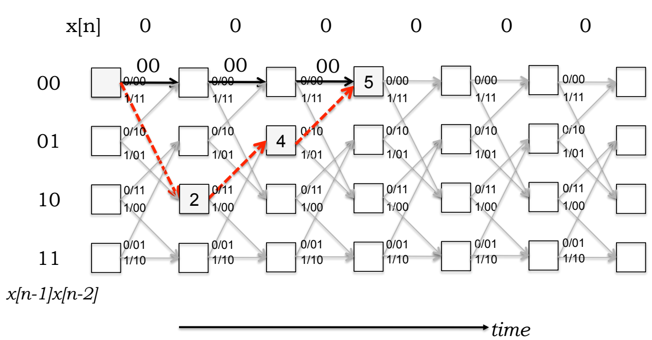


- Soft-decision decoder chooses path that minimizes sum of the squares of the Euclidean distances between received and expected voltages
  - Different BM & PM values, but otherwise the same algorithm

Performance of Viterbi Decoding

- Complexity is linear in message length and *exponential* in *K*, the constraint length
- Code rate: $1/r$
- How to get higher rates or other rates?
  - Answer: *Puncturing*
- How much error correcting capability do we get from a convolutional code?
  - In general, larger values of *K* and *r* (the number of parity streams or generators) provide higher error tolerance
  - But what determines the error correction ability? (I.e., what's the equivalent of the Hamming distance?)
  - Answer: *Free distance*

Free Distance of a Convolutional Code



The free distance is the difference in path metrics between the output when the input is all zeroes, and the output the first input bit along being a '1'. In this example, it is 5 because the first transition outputs '11', the second outputs '11', and the third '10', at which time it converges to the correct state.

Performance (BER) v. SNR for rate-1/2 codes