• Decoding convolutional codes with the Viterbi algorithm
Single Link Communication Model

Original Source

Digitize (if needed)

Source coding

Source symbols

Bit stream

End-host devices

Channel coding (bit error protection)

Mapper + transmit samples

Signals (voltages) over physical link

Receive samples + Demapper

Channel decoding (bit error correction)

Bit stream

Receiving app/user

Source decoding

Render, display, etc.

Bit stream
Single Link Communication Model

first part of course: source and channel coding/decoding

efficiently encoding source symbols as bits, and making the encodings robust to errors

today’s goal: efficiently decode convolutional codes
Convolutional Codes

1. Act on a **stream of message bits** into the encoder, rather than on discrete blocks of data.

2. We **transmit only the parity bits**, not the data and parity bits.

\[ p_0[n] = x[n] + x[n-1] + x[n-2] \]
\[ p_1[n] = x[n] + x[n-2] \]
Trellis View
codeword: 11 10 11
message: 100
received codeword: 11 00 01
path’s codeword: 11 10 11

Hamming distance (110001, 111011) = 2
received codeword: 11 00 01

Hamming distance (110001, 111011) = 2
Hamming distance (110001, 001101) = 4

it's more likely that the correct codeword corresponds to the red path, not the purple path.
\[ p_0[n] = x[n] + x[n-1] + x[n-2] \]
\[ p_1[n] = x[n] + x[n-2] \]

**branch metric**: the distance between the received parity bits and the expected parity bits on each transition

**path metric** \([s,i]\): the smallest sum of branch metrics, minimized over all message sequences that place the transmitter at state \(s\) at time \(i\)
**Viterbi Algorithm**

**For each state s**

1. Determine the predecessor states $\alpha$ and $\beta$

2. Calculate the branch metrics for the transitions $\alpha \rightarrow s$ and $\beta \rightarrow s$ using the received parity bits

3. Determine $PM[s, i+1] = \min(PM[\alpha, i] + BM_\alpha, PM[\beta, i] + BM_\beta)$

4. Keep track of which predecessor state was used

$BM_\alpha = \text{HD}(00, 01) = 1$

$BM_\beta = \text{HD}(11, 01) = 1$

$PM[s, i+1] = \min(1 + 1, 4 + 1) = 2$
Viterbi Algorithm

For each state $s$

1. Determine the predecessor states $\alpha$ and $\beta$
2. Calculate the branch metrics for the transitions $\alpha \rightarrow s$ and $\beta \rightarrow s$ using the received parity bits
3. Determine $\text{PM}[s,i+1] = \min(\text{PM}[\alpha,i] + \text{BM}_\alpha, \text{PM}[\beta,i] + \text{BM}_\beta)$
4. Keep track of which predecessor state was used

The ending state $s$ with the smallest path metric is the most-likely ending state; to recover the message, we trace back the path from state $s$
Binary Symmetric Channel

Encode data  binary symmetric channel  Decode data
Binary Symmetric Channel

Encode data

send data as voltages data

send ‘0’ bit as 0V, ‘1’ bit as 1V

binary symmetric channel

Convert voltages to bits

if voltage ≥ .5, interpret as ‘1’; else, ‘0’

Decode data

Data processing flow:
1. **Encode data**: Convert binary data into voltages.
2. **Send data as voltages data**: Transmit the voltages through the channel.
3. **Convert voltages to bits**: At the receiver end, convert the voltages back into bits.
4. **Decode data**: Interpret the received bits as binary data.

Key rules for encoding:
- ‘0’ bit represented as 0V
- ‘1’ bit represented as 1V

Key rules for decoding:
- If voltage is ≥ 0.5V, interpret as ‘1’.
- Otherwise, interpret as ‘0’.
Free Distance

difference in path metrics between the all-zero output and the path with the smallest non-zero path metric going from 00 to 00

\[ d_{\text{free}} = 5 \]
Summary of 6.02 Part 1

- **Encoding data efficiently** via Huffman codes and LZW compression

- Adding **error-correcting capabilities** to our codes via linear block codes (e.g., Hamming codes) and convolutional codes

- **Decoding efficiently** with syndrome decoding (for linear block codes) and Viterbi decoding (for convolutional codes)