• Convolutional codes
• State-machine view & trellis
Error Control Codes for Interplanetary Space Probes

• Early Mariner probes, 1962-1967 (Mars, Venus) – no ECC

• Later Mariner and Viking probes, 1969-1976 (Mars, Venus) – linear block codes, e.g.,

(32,6,16) bi-orthogonal or Hadamard code
– codewords comprise: the all-0 word, the all-1 word, and the other codewords all have sixteen 0’s, sixteen 1’s. The complement of each codeword is a codeword.
Bi-orthogonal Codes

• e.g., used on Mariner 9 (1971, Mars orbit) to correct picture transmission errors.
  – Data word length: k=6 bits, for 64 grayscale values.
  – Usable block length n around 30 bits. Could have done 5-repetition code, but comparable rate with better error correction from a [32, 6, 16] Hadamard code.
  – Used through the 1980’s.

• The efficient decoding algorithm was an important factor in the decision to use this code. The circuitry used was called the "Green Machine".

• More generally for such codes,
  \[ n=2^{(k-1)}, \quad d=2^{(k-2)} \]
Mariner 9 (400 million km trip)

• “Spacecraft control was through the central computer and sequencer which had an onboard memory of 512 words. The command system was programmed with 86 direct commands, 4 quantitative commands, and 5 control commands. Data was stored on a digital reel-to-reel tape recorder. The 168 meter 8-track tape could store 180 million bits recorded at 132 kbits/s. Playback could be done at 16, 8, 4, 2, and 1 kbit/s using two tracks at a time. Telecommunications were via dual S-band 10 W/20 W transmitters and a single receiver through the high gain parabolic antenna, the medium gain horn antenna, or the low gain omnidirectional antenna.” (NASA)
7329 images, e.g.: 

Image in the public domain. Source: NASA
More powerful codes needed for higher data rates with limited transmitter power

• Space probe may have a 20W transmitter to cover tens of billions of kilometers!
  – Part of the secret is the antenna --- directs the beam to produce the same received intensity as an omnidirectional antenna radiating in the megawatts
  – Also “cryogenically-cooled low-noise amplifiers, sophisticated receivers, and data coding and error-correction schemes. These systems can collect, detect, lock onto, and amplify a vanishingly small signal that reaches Earth from the spacecraft, and can extract data from the signal virtually without errors.” (JPL quote)

• Convolutional codes with Viterbi decoding – Voyager (1977) onwards, Cassini, Mars Exploration Rover, …
Phoning home using a $K=15$, rate=$1/6$ convolutional code
82,950 bps
(Cassini Saturn probe, Mars Pathfinder, Mars Rover)

Image in the public domain. Source: NASA
Convolutional Codes
(Peter Elias, 1955)

• Like the block codes discussed earlier, send parity bits computed from blocks of message bits
  – Unlike block codes, generally don’t send message bits, send only the parity bits! (i.e., “non-systematic”)
  – The code rate of a convolutional code tells you how many parity bits are sent for each message bit. We’ll mostly be talking about rate $1/r$ codes, i.e., $r$ parity bits/message bit.
  – Use a sliding window to select which message bits are participating in the parity calculations. The width of the window (in bits) is called the code’s constraint length $K$.

\[
p_0[n] = x[n] + x[n-1] + x[n-2]
\]

Addition mod 2 (aka XOR)

\[
p_1[n] = x[n] + x[n-2]
\]
Parity Bit Equations

• A convolutional code generates sequences of parity bits from sequences of message bits by a convolution operation:

\[ p_i[n] = \left( \sum_{j=0}^{K-1} g_i[j] x[n-j] \right) \mod 2 \]

• \( K \) is the constraint length of the code
  – The larger \( K \) is, the more times a particular message bit is used when calculating parity bits
    → greater redundancy
    → better error correction possibilities (usually, though not always)

• \( g_i \) is the \( K \)-element generator for parity bit \( p_i \).
  – Each element \( g_i[j] \) is either 0 or 1
  – More than one parity sequence can be generated from the same message; the simplest choice is to use 2 generator polynomials
Transmitting Parity Bits

• We’ll transmit the parity sequences, not the message itself
  – As we’ll see, we can recover the message sequences from the parity sequences
  – Each message bit is “spread across” $K$ elements of each parity sequence, so the parity sequences are better protection against bit errors than the message sequence itself

• If we’re using multiple generators, construct the transmit sequence by interleaving the bits of the parity sequences:

\[ xmit = p_0[0], p_1[0], p_0[1], p_1[1], p_0[2], p_1[2], \ldots \]

• Code rate is $1/$number_of_generators
  – 2 generators $\rightarrow$ rate = $\frac{1}{2}$
  – Engineering tradeoff: using more generators improves bit-error correction but decreases rate of the code (the number of message bits/s that can be transmitted)
Example

• Using two generators:
  - \( g_0 = 1, 1, 1, 0, 0, \ldots \) abbreviated as 111 for \( K=3 \) code
  - \( g_1 = 1, 0, 1, 0, 0, \ldots \) abbreviated as 110 for \( K=3 \) code

• Writing out the equations for the parity sequences:
  - \( p_0[n] = x[n] + x[n-1] + x[n-2] \)
  - \( p_1[n] = x[n] + x[n-2] \)

• Let \( x[n] = [1,0,1,1,\ldots] \); \( x[n]=0 \) when \( n<0 \):
  - \( p_0[0] = (1 + 0 + 0) \mod 2 = 1 \), \( p_1[0] = (1 + 0) \mod 2 = 1 \)
  - \( p_0[1] = (0 + 1 + 0) \mod 2 = 1 \), \( p_1[1] = (0 + 0) \mod 2 = 0 \)
  - \( p_0[2] = (1 + 0 + 1) \mod 2 = 0 \), \( p_1[2] = (1 + 1) \mod 2 = 0 \)
  - \( p_0[3] = (1 + 1 + 0) \mod 2 = 0 \), \( p_1[3] = (1 + 0) \mod 2 = 1 \)

• Transmit: 1, 1, 1, 0, 0, 0, 0, 1, ...
Shift-Register View

- One often sees convolutional encoders described with a block diagram like the following:

The values in the registers define the state of the encoder.

- Message bit in, parity bits out
  - Input bits arrive one-at-a-time from the left
  - The box computes the parity bits using the incoming bit and the \( K-1 \) previous message bits
  - At the end of the bit interval, the saved message bits are *shifted right* by one, and the incoming bit moves into the left position.
Example: Transmit message 1011

\[ p_0[n] = x[n] + x[n-1] + x[n-2] \]
\[ p_1[n] = x[n] + x[n-2] \]

Xmit seq: 1, 1, 1, 0, 0, 0, 0, 1, ...
(codeword)
State-Machine View

- Example: $K=3$, rate-$\frac{1}{2}$ convolutional code
- There are $2^{K-1}$ states
- States labeled with $(x[n-1], x[n-2])$ value
- Arcs labeled with $x[n]/p_0[n]p_1[n]$
- msg=101100; xmit = 11 10 00 01 01 11

$p_0[n] = x[n] + x[n-1] + x[n-2]$
$p_1[n] = x[n] + x[n-2]$

(Generators: $g_0 = 111$, $g_1 = 101$)

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.
Trellis View

• State machine unfolded in time (fill in details using notes as guide, for the example considered here!)

\[
\begin{array}{cccc}
00 & 00 & 00 & 00 \\
01 & 01 & 01 & 01 \\
10 & 10 & 10 & 10 \\
11 & 11 & 11 & 11 \\
n=0 & n=1 & n=2 & n=3 & \ldots
\end{array}
\]
The Parity Stream forms a Linear Code

• Smallest-weight nonzero codeword has a weight that (locally in time) plays a role analogous to \( d \), the minimum Hamming distance. It’s called the free distance (\( fd \)) of the convolutional code.

• What is \( fd \) for our example?
Encoding & Decoding Convolutional Codes

• Transmitter (aka Encoder)
  – Beginning at starting state, processes message bit-by-bit
  – For each message bit: makes a state transition, sends $p_0 p_1 \ldots$
  – Pad message with $K-1$ zeros to ensure return to starting state

• Receiver (aka Decoder)
  – Doesn’t have direct knowledge of transmitter’s state transitions; only knows (possibly corrupted) received parity bits, $p_i$
  – Must find most likely sequence of transmitter states that could have generated the received parity bits, $p_i$
  – If BER < $\frac{1}{2}$, $P$ (more errors) < $P$ (fewer errors)
  – When BER < $\frac{1}{2}$, maximum-likelihood message sequence is the one that generated the codeword (here, sequence of parity bits) with the smallest Hamming distance from the received codeword (here, parity bits)
  – I.e., find nearest valid codeword closest to the received codeword – Maximum-likelihood (ML) decoding
In the absence of noise ...

• Decoding is **trivial**:

\[
p_0[n] = x[n] + x[n-1] + x[n-2]
\]

\[
p_1[n] = x[n] + x[n-2]
\]

• Can you see how to recover the input \(x[.\,]\) from the parity bits \(p[.\,]\) ?

• In the presence of errors in the parity stream, message bits will get corrupted at about the same rate as parity bits, with this simple-minded recovery.
Spot Quiz!

Consider the convolutional code given by

\[ p_0[n] = x[n] + x[n-2] + x[n-3] \]
\[ p_1[n] = x[n] + x[n-1] + x[n-2] \]
\[ p_2[n] = x[n] + x[n-1] + x[n-2] + x[n-3] \]

1. Constraint length, \( K \), of this code = 

2. Code rate = 

3. Coefficients of the generators = , , 

4. No. of states in state machine of this code = 
