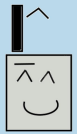
FT8 and Error Correction

Jeff Kabel AA6XA

About Me

- BS and MS in Electrical Engineering, University of Rochester
- Ham since 2004, Extra since 2007
  - AA6XA, formerly KB1KXL
- Ham radio interests include
  - SOTA
  - VHF contesting
  - Homebrewing
  - Microwaves
  - HF CW contesting
- Social Media
  - ham_bitious (youtube.com/@hambitious and Instagram, Mastodon, etc.)



Outline

- What is Error Correcting?
- Simple FEC Block Codes
  - 3x Repeat Code
  - Hamming(7,4) Code
- FT8 Error Correcting
  - LDPC Codes

What is Error Correction?

- If we send a message over a noisy channel, it is possible the message will be corrupted by noise (or fading, etc.)
  - Perhaps you sent a “D” in CW, but due to noise it was received as an “N”
  - On sideband you say “B” but the other station hears “D” or “P”
  - In a digital mode, send 0101 but receive 0111
- It would be nice if we could detect when this happens, and even better if we could fix errors without asking for a repeat
  - This is Forward Error Correction
  - The other choice is to ask for repeats

Simple Forward Error Correction

- In voice modes we can use the phonetic alphabet
  - “Bravo” sounds different than “Delta” or “Papa”
  - Doesn’t work for CW
- Repeat the message a few times
  - This works, but is rather inefficient. Also is not FEC
- With digital messages we can use more powerful techniques
  - Block Codes - FT8 uses this
  - Convolutional Codes

Parity Check Bit

- What is a parity bit?
  - Bit that is set so the number of 1’s is even
  - Equivalently, sum of the bits (mod 2) = 0
- Examples
  - 010x -> 0101
  - 101x -> 1010
- Block codes work by adding parity bits

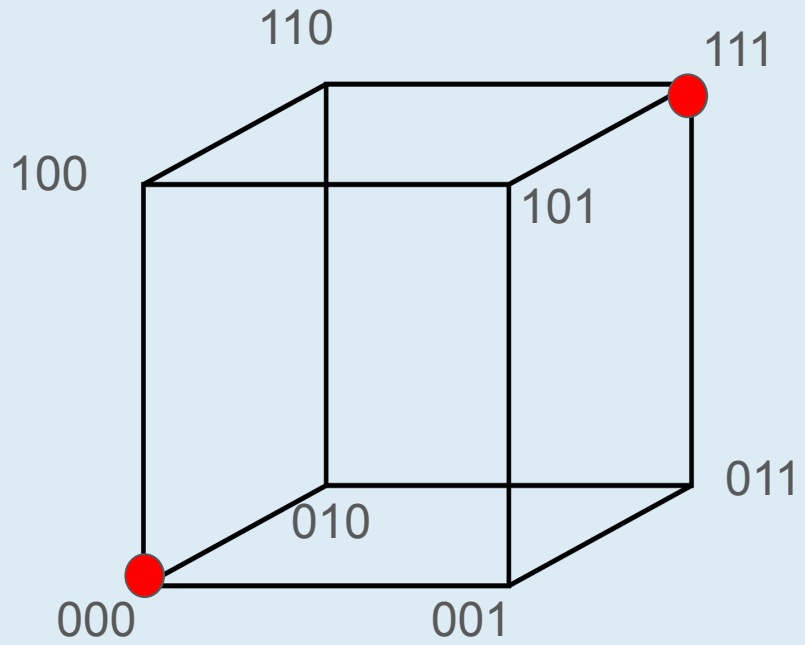
Linear Block Codes

- Code that operates on blocks of data
  - Versus a convolutional code, which operates on streams of data
- Block codes have certain parameters
  - Message length (k)
  - Block length or Codeword length (n)
  - Distance (d)
  - Rate (R)
- These parameters determine how many errors can be detected or corrected
  - Detect d-1 errors, or correct *floor*(d-1 /2) errors
- Block codes are referred to by two numbers, the number of bits in each block, and the number of bits they output
  - e.g. (3,1) or Hamming(7,4)
- In general, can be defined by a generator matrix **G** or parity check matrix **H**

3x Repeat Code

- Simply send each bit three times, e.g. send “111” instead of “1”
- n = 3, k = 1, so we call this a (3,1) code
- R = 1/3
- This is not very efficient, 2/3 of the data we send is error correction, and we can only fix one error (d = 3) or detect two
- Decode by “majority vote”
    - [1 1 0] or [1 0 1] or [0 1 1] are obviously supposed to be [1 1 1]

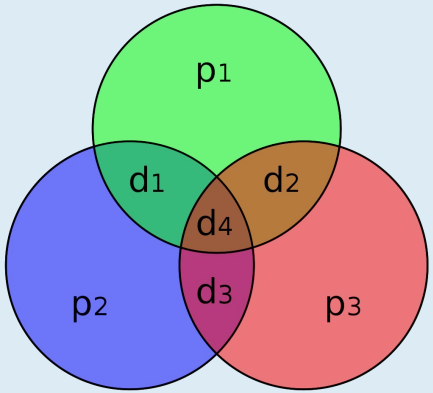
3x Repeat Code



- n=3
- k=1
- R = 1/3
- d=3
  - Correct 1 or detect 2 errors

Hamming(7,4) Code

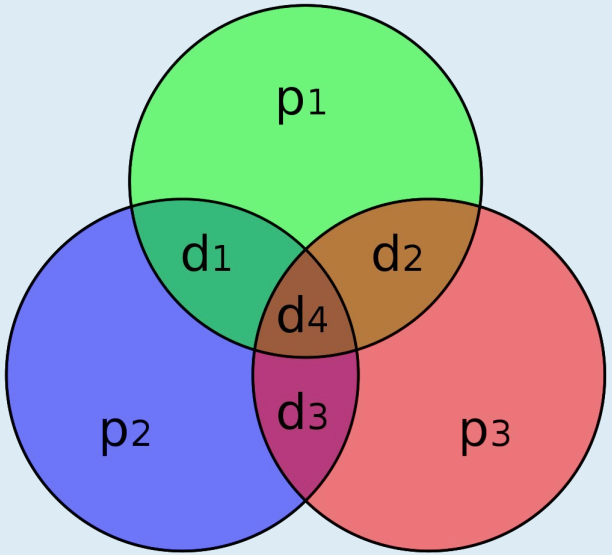
- This code was the first error correcting code invented (1950 or so)
- Can correct one error and detect two
- Adds three parity check bits to four bits of data
- R = 4/7 = 0.571
  - This is a lot better than the 3x repeat code with R = 1/3 = 0.333



Hamming(7,4) Encoding Example

- m = [1 0 0 1]
- p1 is taken from 1st (1), 2nd (0), 4th (1) data bits
  - Already even parity, so p1 = 0
- p2 is from 1st (1), 3rd (0), 4th (1)
  - p2 = 0
- p3 is from 2nd (0), 3rd (0), 4th (1)
  - p3 =1
- C = [1 0 0 1 0 0 1]

- m = [1 0 1 1]
- C = [1 0 1 1 0 1 0]



Hamming(7,4) Encoding

- If you're comfortable with linear algebra, this can be represented more compactly with matrix multiplication
-

$$m = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$$

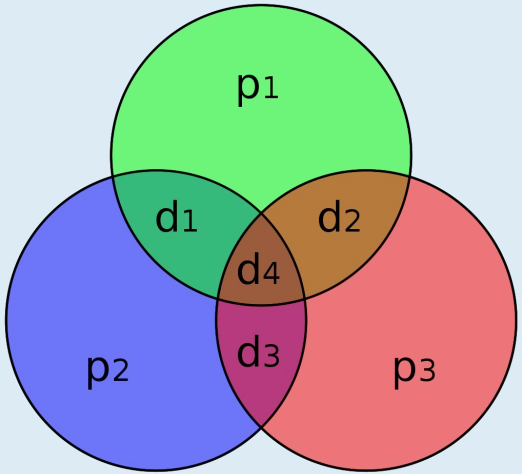
$$C = \mathbf{m}G$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

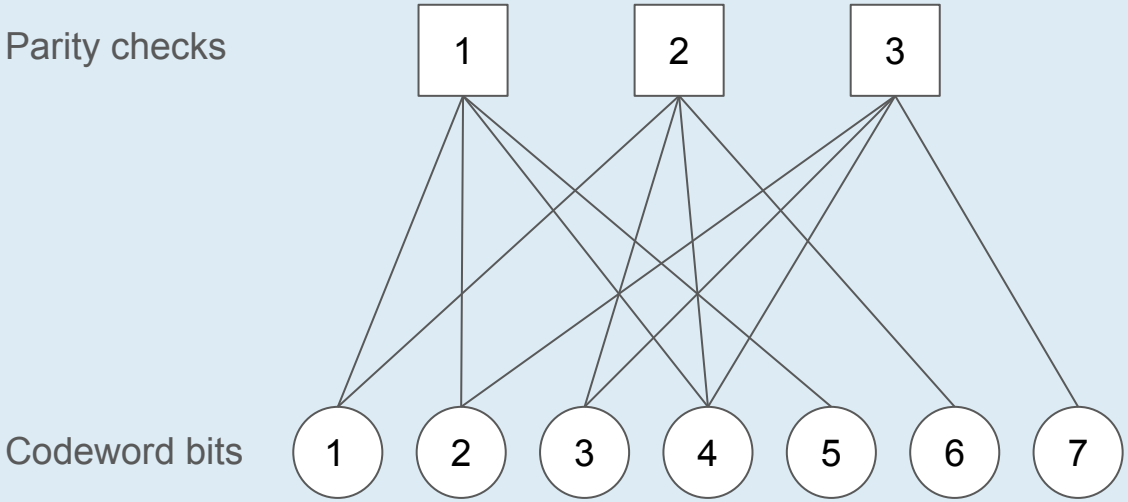
$$C = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

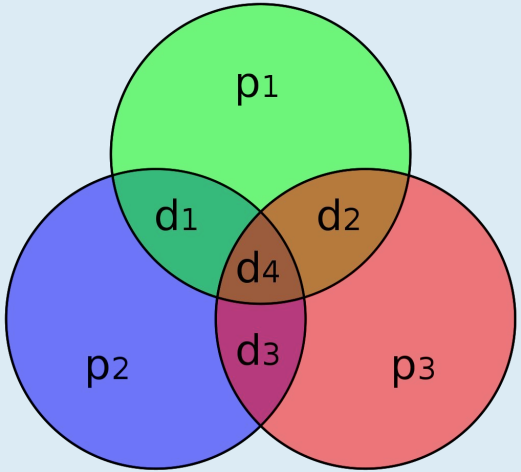
Hamming(7,4) Decoding Example

- This code is harder to decode by hand than the 3x code
- Check the parity of each of the parity bits, figure out which, if any, bits have been flipped
- Example: r = [1 0 0 1 0 0 1]
  - No errors!
- Example: r = [1 1 0 1 0 0 1]
  - p1 & p3 fail, so d2 must be wrong
- Example: r = [1 0 0 1 1 0 1]
  - Only p1 fails, so it must be wrong
- We could try and draw a cube like the 3x code, but it would need to be in 7 dimensions!
  - I am not able to do that, sorry



Hamming(7,4) Tanner Graph





Syndrome Decoding

- With a little linear algebra, decoding this gets much easier
- Compute the syndrome $s = Hr^T$
  - If the answer is 0, there are no errors
  - Otherwise, the syndrome will be a column from the parity check matrix and the column number will be location of the error
- Example:

$$
\begin{aligned}
r &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \\
s &= Hr^T \\
&= \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T \\
&= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

Another Example

$$
\begin{aligned}
r &= \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \\
s &= Hr^T \\
&= \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T \\
&= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
\end{aligned}
$$

This can be done with the 3x repeat code as well!

Decoding Algorithms

- There are a number of other decoding algorithms used, but describing them is beyond this presentation
- As blocks get bigger with more error correcting, simple algorithms start to take a lot more time. More clever ways are needed to decode

Other Block Codes

- Higher order Hamming codes
- Hadamard Codes
  - Used on the Mariner 9 probe to send images back from Mars
- Golay Codes
  - Used on the Voyager probes in the outer Solar system
- Reed-Solomon Codes
  - Used on CDs, DVDs, QR Codes, and more
- Low Density Parity Check codes
  - Used in FT8

Back to FT8!

- We now know what Forward Error Correction is
- From the WSJT-X manual: “Forward error correction (FEC) in FT8 uses a low-density parity check (LDPC) code with 77 information bits, a 14-bit cyclic redundancy check (CRC), and 83 parity bits making a 174-bit codeword. It is thus called an LDPC (174,91) code. Modulation is 8-tone frequency-shift keying (8-GFSK) at 12000/1920 = 6.25 baud. Synchronization uses 7×7 Costas arrays at the beginning, middle, and end of each transmission. Transmitted symbols carry three bits, so the total number of channel symbols is 174/3 + 21 = 79. The total occupied bandwidth is 8 × 6.25 = 50 Hz”

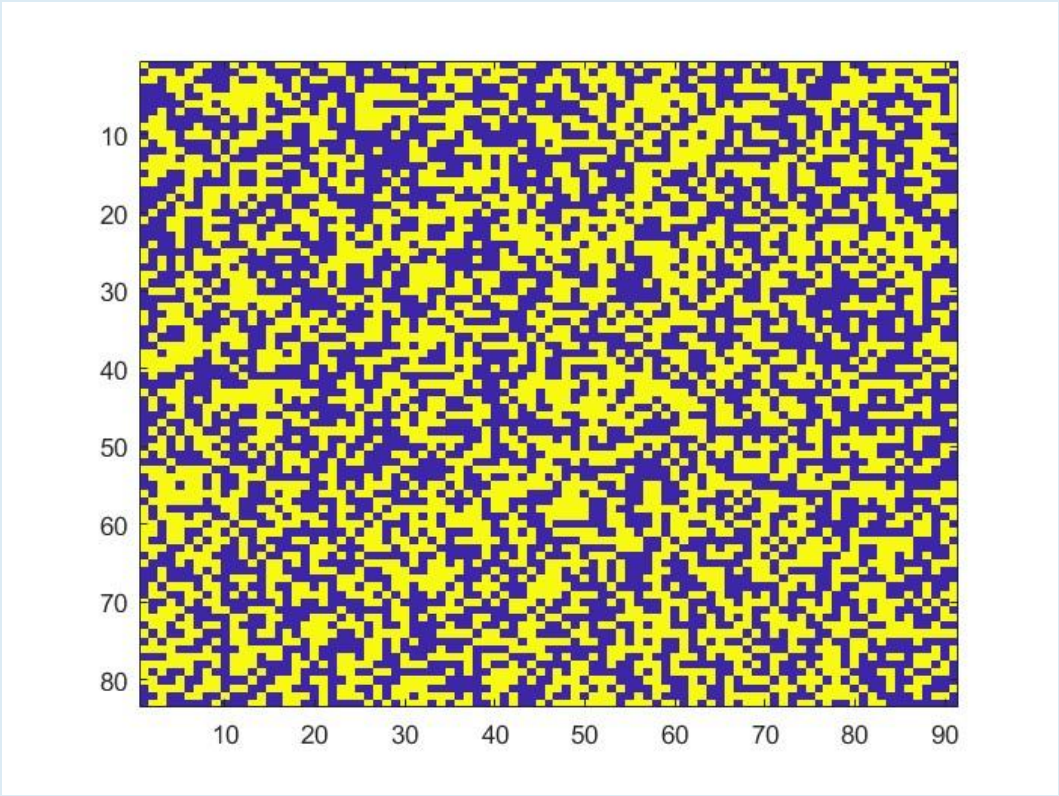
Low Density Parity Check Code

- The LDPC code was invented in the early 1960's by Robert Gallager in his doctoral dissertation
  - At the time they were too computationally expensive to actually use
  - By the 90's computers had become powerful enough engineers started to take another look
- The LDPC code is a type of linear block code that uses a sparse parity check matrix
  - A sparse matrix is a matrix that is mostly zeros
- LDPCs are some of the best error correcting codes we have
  - They approach the Shannon-Hartley limit, which gives the maximum rate information can be transmitted in a given channel

FT8 LDPC

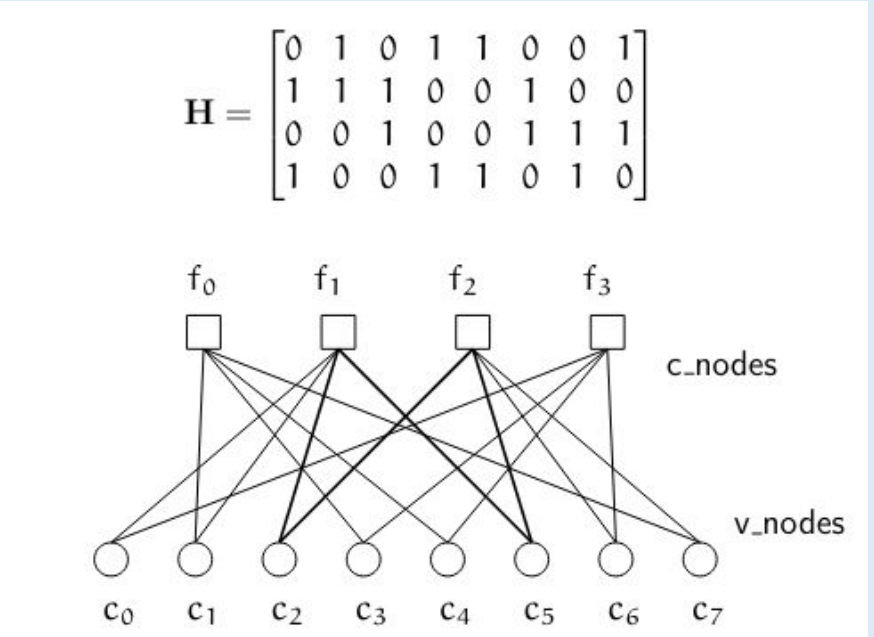
- FT8 uses a LDPC(174,91) code
  - This means the generator matrix is 91 rows x 174 columns, or that 91 information bits becomes 174 codeword bits (83 error correcting bits)
  - The parity check matrix is also huge
- If you want to see the generator and parity check matrices, they’re in the WSJT-X source code
  - *ldpc_174_91_c_generator.f90* and *ldpc_174_91_c_parity.f90* files respectively

FT8 Generator Matrix

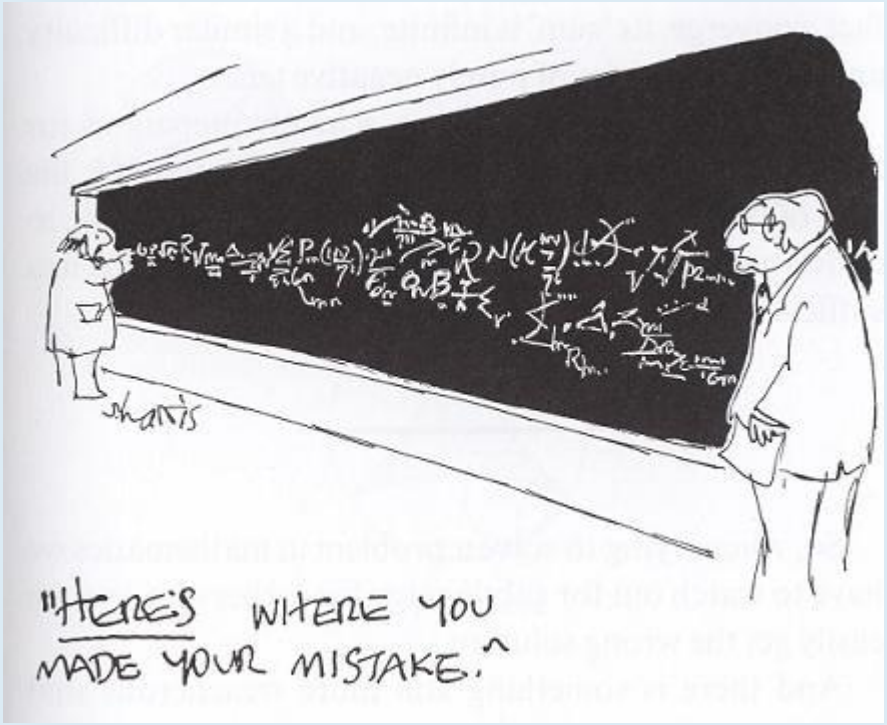


FT8 LDPC Decoding

- FT8 uses an iterative decoder
- With so many parity checks, changing one bit will have a big ripple effect, so iterating is the best way to decode



Questions?



Slides:

Further Reading

- https://web.mit.edu/6.02/www/s2012/handouts/6.pdf
- https://www.eecs.umich.edu/courses/eecs555/chap7.pdf
- https://uweb.engr.arizona.edu/~ece506/readings/project-reading/5-ldpc/LDPC_Intro.pdf
- https://cmrr-star.ucsd.edu/static/presentations/ldpc_tutorial.pdf