Coding Theory and Hamming Codes (10/12/2020)

Coding theory is about sending the most accurate messages. We want to send a string of bits to someone. We don't care about it being intercepted by someone else, etc. (this isn't cryptography!), but there might be errors in transmitting the code! Thus, we really want to be able to catch errors in the code and fix them if they have been made.
For now, we are assuming that there are exactly zero or one errors in any message. We want to be able to "encode" and send the message. While sending, the message will either stay the same or have one error. Then, the receiver receives the message with the possible error.

Model:
Alice (A) will be sending a message to Bob (B):

$$M(\text{message}) \to C_A(\text{encoded message}) \to C_B(\text{encoded message with error})$$

Bob will receive C_B, Alice will send C_A.

Attempt at a solution (2-times):
We can try and just send two of each bits, i.e. $(1,0,0,1) \to (1,1,0,0,0,0,1,1)$. This will allow us to detect errors, but how do we correct them! If we get back $(1,0)$ or $(0,1)$, Bob knows the message is corrupted, but doesn't know if it is $(1,1)$ or $(0,0)$.

Attempt at a solution again! (3-times):
If we send each bit three times, we can try encoding our message again, i.e. $(1,0,0,1) \to (1,1,1,0,0,0,1,1,1)$. This will work! Because there can only be one error, we can both a) detect the error (if one is different than the other two each triplet of bits) and b) fix the error (because the error will be the odd one out each triplet of bits).
We define *redundancy* as the number of bits in the codeword divided by the number of bits in the original message. We see that 3-times algorithm has a redundancy of 3! This is pretty memory heavy. Let's see if we can make something lighter!

The (7,4) Hamming Code (Our Best Solution):
Realize that for all $m \geq 2$, there is a $(2^m - 1, 2^m - m - 1)$ Hamming code. The codeword length will be $2^m - 1$, and the message length will be $2^m - m - 1$. Thus, if $m = 3$, we get a $(7,4)$ Hamming code (the code we'll be working with)!
Valid codewords are elements of the nullspace of H:

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} F \mid I_3 \end{pmatrix}$$

We are thinking about everything in binary! This means everything is modulo 2, that is 2=0, -1=1, etc. Thus, the basis of $N(H)$ is going to be columns of:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

We encode a message M as simply $GM = C_A$, while will produce a linear combination of the columns of G, and thus of the basis of $N(H)$. Accordingly, if there are no errors, $H(C_B) = \vec{0}$.

If there is an error, then we can find it in one of two ways. Secondly, we can use the columns of H to figure out where the mistake is. Think, a message with an error, C_B, is equal to $C_A + e_i$, where the error is at position i and e_i is a standard basis vector. It follows that:

$$HC_B = H(C_A + e_i) = HC_A + He_i = \vec{0} + He_i = He_i = H_{*i}$$

Thus, the error in C_B will be at the corresponding number of the column of H that is equal to to HC_B.
It is important to note that as the top four rows of G are I_4, we know that the top four elements of C_A or C_B are the message itself, and the bottom three are a parity check. Thus, messages are of the form:

$$C = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Finally, note that for a message of four bits, we are sending an encoded message of seven bits. Thus, our final redundancy is $7/4 = 1.75$. Much better than 3-times!

Hamming Code in General:
Define the $(2^m - 1, 2^m - m - 1)$ Hamming code. First, pick an m. The resulting parity check matrix H will be:

$$H = (F \mid I_m)$$

Note that the columns of H will have no all-zero columns and no two columns that are the exact same. Now, our generator matrix will be:

$$G = \left(\frac{I_{2^m-m-1}}{-F}\right) = \left(\frac{I_{2^m-m-1}}{F}\right)$$

Note that as we are working in binary, $-F = F$.
To generalize our process, for Alice to transmit message M to Bob:

- Alice encodes M as $C_A = GM$.
- Alice sends the code C_A. While sending, C_A gets one or zero errors, becomes C_B.
- Bob receives C_B, runs a parity check by finding HC_B, and corrects based on those results.

This works out really well! The only problem is that more than one error will break the system. Finally, note that the redundancy of our system will always be $\frac{2^m-m-1}{2^m-1}$. As m gets bigger and bigger, this value approaches 1! This means for a near-infinite m, it takes almost the same amount of memory to send the message itself as it takes to send the encoded message!