CSC 310, Spring 2002 — Assignment #4

Due at **start** of tutorial on April 12. Worth 10% of the course grade.

Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own.

In this assignment, you will empirically investigate the performance of three decoding algorithms for the product of two $[7, 4]$ binary Hamming codes, which is a $[49, 16]$ code.

A codeword in this product code can be visualized as a 7×7 array of bits, which can be converted to a vector of length 49 by a raster scan across and down the array. By the definition of a product code, the bits in every row and in every column of a codeword for the product code will be codewords of the $[7, 4]$ Hamming code. The product code can be used in a systematic form in which we encode a block of 16 message bits by first placing these bits in the 4×4 upper-left portion of the 7×7 array. The check bits for each of these four rows can then be computed by the usual method for a Hamming code, at which point the whole of the first four rows will be filled in. The last three rows are then filled in by computing the three check bits for each column.

You will investigate the following methods for decoding a 7×7 block of received bits sent through a BSC:

1. Apply the nearest-neighbor decoding method for the $[7, 4]$ Hamming code to each row of the received block, replacing each row with its decoding. Then apply the nearest-neighbor decoding method for the Hamming code to each column of this modified block, replacing each column by its decoding.

2. Same as above, but after applying the Hamming code decoding method to each row and then each column, go back and apply it to each row again, and then each column again.

3. Decode to a nearest neighbor of the received 7×7 block by considering all 2^{16} codewords, picking one that has minimum Hamming distance from the received block.

You can implement the last method by counting through the integers from 0 to $2^{16} - 1$. For each such integer, place the bits of its binary representation in the 4×4 upper-left portion of a codeword, and then fill in the rest of the 7×7 array as is done for encoding a message block. Find the Hamming distances of these codewords from the received data, and pick one that has the smallest distance (resolving ties arbitrarily).

You should find out how well these decoding methods work by simulating the transmission of 10000 random messages through a BSC with error probability 0.08. For each of these 10000 messages, you should do the following:

1. Randomly generate a block of 16 messages bits, with each bit being equally likely to be 0 or 1, independently.

2. Encode this block as a 49-bit codeword of the product code, as described above.

3. Simulate transmission of this codeword through the channel by flipping each bit with probability 0.08, independently for each bit.

4. Decode this noisy version of the codeword by each of the three methods described above.

5. Determine whether each of these three decodings matches the transmitted codeword in all 49 bit positions, and also whether it matches in the 16 message bit positions.

For the random decisions in steps (1) and (3), you can use the `drand48` procedure in the C library. This procedure returns a (pseudo) random real value uniformly distributed between 0 and 1. A random bit that is 1 with probability p can be produced as follows:

```
bit = drand48() < p;
```

Include `<stdlib.h>` at the beginning of your program to declare `drand48`.

The output of your program should be a table showing how often each of the three decoding methods produced the correct codeword, and how often it produced the correct message bits (even if some of the check bits were not correct). You should print such statistics based on all 10000 messages, and also give these statistics for the subsets of messages in which there where 0 errors, 1 error, 2 errors, and so forth, up to 9 errors, plus a category for 10 or more errors. (Also print the number of messages in each such category, so the reliability of the figures can be determined.)

You should hand in a listing of your program, which should be written in reasonable programming style, but which you do **not** need to write in a way that could easily be generalized. You should also hand in the output of your program, and a discussion of the results, commenting on the relative performance of the three methods, and on anything else of interest that you have noticed.