CSC310 – Assignment #4

Due: Dec. 4, 2006, 10am at the **START** of tutorial or by email beforehand
Worth: 17%
Late assignments not accepted.

1 Low Density Parity Check Codes over the Binary Erasure Channel

In this assignment you will implement an encoder and decoder for sending messages over a binary erasure channel using a Low Density Parity Check (LDPC) Code.

- From the course website, download the two matrices **G** and **H** which represent the generator matrix (in systematic form) and the parity check matrix for a simple linear LDPC code. The code encodes messages of length 16 bits by transforming them into 32 bit long codewords, and hence has rate 1/2. Each of the 16 parity check equations depends on exactly 6 of the codeword bits and each of the 32 codeword bits is involved in exactly 3 parity check equations. The matrices **G** and **H** are shown below.

  G

  ```
  10000000000000000000101001000000
  01000000000000001100100111001000
  00100000000000001000010011110100
  00010000000000001111011110111101
  00001000000000001011111100011001
  00000100000000001001100000000000
  00000010000000000000000000000111
  00000001000000000100111000110100
  00000000100000000111011000111010
  00000000010000000001101100011010
  00000000001000001010001100001000
  00000000000100001111011100010001
  00000000000010000111111111011011
  00000000000001000111100001111100
  00000000000000100001101000010111
  00000000000000010011000000001000
  ```

  H

  ```
  00110100001100001000000000000000
  01000100000000110101000000000000
  01001000010100000010000000001000
  00001001001000010110000000000000
  00000010000000001000101000010001
  00001000000101000000010000000110
  10000010000000000000001100100100
  00000011001000100000000100000001
  00000000000000000001100111001000
  10000000000001000000010001000011
  01110000010010000000000010000000
  00100101000000010001000000010000
  00000000110011000000000010100000
  10010000100010000000000001100000
  00000000100000001100111000000000
  00000000000001000100000000011110
  ```

- Encoding using this code is very simple: you multiply your 16 bit source message by the generator matrix G (and take the result mod 2).

- Decoding is more complex. For the binary erasure channel (BEC), you will use the following procedure: Begin with the received codeword, which has either a 0, a 1 or a ? in each position. Until there are no more ? left, find one of the 16 checks for which exactly one of the 6 codeword bits it depends on is a ?. (If no such check exists, decoding terminates in failure.) Use that check to "fill in" the value of the ? bit by setting the ? to either 0 or 1 so that the check is satisfied. Repeat until all codeword bits have been recovered or until we cannot fill in any more bits. If all the ? have been filled in when we stop, we have recovered the codeword with complete certainty; otherwise we may have filled in some bits but still cannot be sure of some others.

2 What to do and what to hand in

- Implement an encoder for this code. (This is trivial.)
- Implement the iterated message passing decoder. (This is a bit harder.)
- Download the 10000 random messages of length 16 bits each from the course website.
These were generated by setting each bit to 0 or 1 randomly with probability 1/2.
Encode these messages using **G**.
- Using the program `/u/roweis/public/bec` on CDF, simulate transmitting the resulting codewords over a BEC. This program takes a single command line argument which is the probability f of erasure, reads a string of '0's and '1's from standard input and emits a string of '0's, '1's and '?'s to standard output.
- *It is very important that you run the command from your own account, since its random seed is computed based on your username. Thus, you will not get the same results as anyone else in the class.*
- Run your decoder on each of the received codewords, and measure how many of the original 10000 messages you were able to successfully decode with complete certainty. Do this for BEC erasure probabilities f equal to 0.1,0.2,0.3,0.4 and 0.5; report your results for each noise level.
- Also measure what fraction of the original message bits are correctly recovered. To do this, look at the state of your decoder when it converges, and look only at the codeword positions corresponding to the original message bits. Report your results for each noise level.
- Download the two corrupted transmissions `r1` and `r2` from the course website. In those files, the character ? represents the erasures introduced by the channel. Decode them and extract the message bits from the recovered codewords.. How many of the 575 blocks of `r1` were you able to recover with complete certainty? How many of the 600 blocks of `r2` were you able to recover with complete certainty?
- The message corresponding to `r1` is actually a binary image. Figure out how to display it, and hand in a picture of the image which represents your best guess at the decoding. (Replace any bits you were not able to decode with zeros.)
- The message corresponding to `r2` is actually an ASCII string. Figure out how to map the decoded message bits into ASCII and hand in a printout of your best guess at the decoding. (Replace any ASCII characters whose block who were not able to completely decode with the character '@'.)
- [Hard]. For each noise level above, try to bound how many messages (out of 10000) a *maximum likelihood* decoder would have been able to decode with certainty by taking messages your iterative decoder failed on, finding a remaining parity check that has only 2 of its 6 bits unknown, and setting one of those bits to both 0 and 1. For each setting, continue decoding from that point. If continued decoding converges for *both* settings then you can conclude that no decoder could ever have been absolutely certain of the correct decoding the two settings. If continued decoding converges for one setting but not the other, it is *possible* that a ML decoder would have correctly solved the message. If continued decoding converges for neither setting, you can't conclude anything from that parity check.