CSC 310, Fall 2011 — Practical Assignment #2

Due in class on December 7. 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 write a program that creates the syndrome decoding table for a linear code for a BSC, and from it computes the probability of decoding error for given values of the channel error probability. You will try out this program on some randomly-generated codes whose parity-check matrices I have supplied on the web page, and discuss the results.

Syndrome decoding is described in the lecture notes. For this assignment, you should include in the table an entry for a syndrome of 0 (ie, for when the received vector is a codeword), which will be associated with an "error pattern" of no errors.

You may use any programming language you wish for this assignment, though interpreted languages and languages that do not give access to bit operations aren't recommended, since they might lead to your program not running in a reasonable time for the codes supplied.

Part A (40 marks): For this part, you will implement a module that builds a syndrome decoding table, given the parity-check matrix for a linear code. For the later parts of this assignment, all that is needed from this table is the weight of the error pattern associated with each syndrome, but for ease of debugging, implementing an option to also record the actual error pattern is recommended. (This option should probably not be enabled for actual use in Part B.)

The parity-check matrix specifying a code is stored in a text file whose first line contains the number of rows and number of columns in the parity-check matrix, with this followed by one line per row of the parity-check matrix, each such line containing 0s and 1s separated by spaces.

I suggest that you store a parity-check matrix as an array of 32-bit integers, with each such integer containing the bits in a column of the parity-check matrix. I suggest that you store an error pattern as an array of indexes of positions where errors occur. This will allow you to compute the syndrome for an error pattern as the bit-by-bit exclusive-or (equivalent to addition modulo 2) of the columns in the parity-check matrix corresponding to positions where an error occurred. (In C, the bit-by-bit exclusive-or operator is "^".) If you keep the indexes of error positions in ascending order, it is also fairly easy to move from one error pattern of a given weight to the next.

To allow this representation, you may limit the number of rows in a parity-check matrix to no more than 30, and limit the number of columns to no more than 120. This allows a column of the parity-check matrix to be stored as a single 32-bit word (with two bits to spare, avoiding the need to worry about sign bits). It also allows a column index to be stored in one byte (again, with bits to spare so that sign bits won't be an issue).

You should hand in your program code for this module, in the form in which it is used in Part B. You need not hand in any test output, though I recommend testing this module separately before trying to use it in Part B.

Part B (30 marks): For this part, you will use the weights of the error patterns in the syndrome decoding table to compute the probability that syndrome decoding will lead to the wrong codeword. You should compute this for BSC error probabilities of 0.01, 0.02, ..., 0.14, 0.15, outputting a table giving the probability of decoder error for each such channel error probability.

You should also output a table giving the number of entries in the syndrome table with each error pattern weight — ie, you should say how many syndrome table entries are associated with a single error, how many with two errors, etc. (There will always be a single entry (for a syndrome of zero) associated with no errors.) The information in this table is actually all you need to compute the error probability, which is just one minus the probability that one of these correctable error patterns occurs.

It will be informative to also output the total number of error patterns with each weight, for comparison with the number associated with some entry in the syndrome table.

You can use the `pchk-5-20-1` test file to test your program, for which I have provided the output I obtained (which I hope is correct!). You should hand in your program code (except what you are handing in anyway for Part A).

Part C (30 marks): Finally, you should try out your program for part B on a set of codes (given by parity-check matrices) that I have supplied on the course web page. These parity-check matrices were randomly generated by independently setting each entry to 0 or 1, with equal probabilities. (This procedure is not guaranteed to produce parity-check matrices in which the rows are linearly independent, but it did for these particular matrices.)

The following parity-check matrices are supplied for these tests:

File name	Rows	Columns	Rate	Seed
pchk-10-25-1	10	25	0.6	1
pchk-10-25-2	10	25	0.6	2
pchk-10-25-3	10	25	0.6	3
pchk-12-30-1	12	30	0.6	1
pchk-12-30-2	12	30	0.6	2
pchk-12-30-3	12	30	0.6	3
pchk-14-28-1	14	28	0.5	1
pchk-14-28-2	14	28	0.5	2
pchk-14-28-3	14	28	0.5	3
pchk-16-32-1	16	32	0.5	1
pchk-16-32-2	16	32	0.5	2
pchk-16-32-3	16	32	0.5	3
pchk-17-34-1	17	34	0.5	1
pchk-17-34-2	17	34	0.5	2
pchk-17-34-3	17	34	0.5	3

You may ignore the parity-check matrices with 17 rows if your program takes too long when run on them.

You should hand in the output of your program when run on these parity-check matrices, and discuss what you can conclude from it. Topics you may want to address include how variable is the performance of codes produced with different random seeds, how increasing the size of the parity-check matrix for a given rate affects decoding error probability, how rate affects decoding error probability, how the decoding error probability compares with the probability of not receiving all message bits correctly if they were sent unencoded, and whether a code that is better than another for one value of the channel error probability will also be better for a different channel error probability.