EECE 580B Modern Coding Theory
WWW: http://dde.binghamton.edu/filler/mct/hw/6

Homework assignment #6
Due date: November 10, 2:50pm

Communication over the BEC using LT and LDPC codes

In this assignment, you will compare the performance of 4 different codes (2 LT and 2 LDPC codes) over the Binary Erasure Channel (BEC). We will use a short block-length of $n = 2000$ bits in order to avoid performance implementation issues. Use the following Matlab function to simulate the BEC:

```
function [ y ] = bec_channel( x, e )
%BEC_CHANNEL Simulates binary erasure channel with erasure probability e
  y = x;
  y( rand(size(x))<e ) = -1; % -1 = erasure, otherwise 0s and 1s are bits
end
```

The central problem of this assignment is the implementation of the "peeling solver" which will be used for decoding of both codes.

Problem 1: (Peeling solver)

Let A be a binary matrix of size $m \times l$ and b be a binary vector of size $m \times 1$. Let vector $x \in \{0,1,-1\}^l$ of size $l \times 1$ represents a partial solution of the system $Ax = b$, where $x_i = -1$ means that variable x_i is unknown. Under the assumption that at least one solution exists, implement a Matlab function "peeling_solver.m" that, for given A, b, and x, tries to solve the system of linear equations

$$Ax = b$$

in binary arithmetic only by using a back substitution. If, in some step, the system cannot be solved, output the solution with the least number of unknown variables (set $x_i = -1$ if the variable x_i remains unknown).

I expect the function in the following form:

```
function [ x_hat ] = peeling_solver(A, x, b)
%PEELING_SOLVER tries to solve A*x=b for unknown variables x_i==-1 by using a back
substitution only.
  x_hat = x; % start with this solution
  % and start solving the system
end
```

Problem 2: (Generator matrix of LT codes)

An LT encoder can produce an endless stream of bits, while encoding a k-bit vector of information bits s. If we truncate this encoder after it produces n output bits, we can represent this linear code by a $k \times n$ generator matrix G. This matrix contains random binary columns with the number of ones following the robust soliton distribution $R(x) = \sum_{i=1}^{k} R_i x^i$. (Here, R_i is the relative number of columns in G with i ones.) The robust soliton distribution $R(x)$ is obtained as a composition of two degree distributions $\rho(x) = \sum_{i=1}^{k} \rho_i x^i$ and $\tau(x) = \sum_{i=1}^{k} \tau_i x^i$,

$$R(x) = \frac{\rho(x) + \tau(x)}{\rho(1) + \tau(1)}.$$

The degree distribution $\rho(x)$ is called the ideal soliton distribution and contains the following coefficients

$$\rho_1 = \frac{1}{k} \text{ and } \rho_i = \frac{1}{i(i-1)} \text{ for } i = 2, \dots, k.$$

The degree distribution $\tau(x)$ is used to increase the average number of columns with exactly one 1 and thus makes the decoding algorithm successful. This degree distribution depends on two parameters c and δ and contains the following coefficients

$$\tau_i = \frac{S}{k}\frac{1}{i} \text{ for } i = 1, \dots, \lfloor k/s \rfloor, \quad \tau_i = \frac{S}{k}\ln(S/\delta) \text{ for } i = \lceil k/s \rceil, \text{ and } \tau_i = 0 \text{ for } i > \lceil k/s \rceil,$$

where $S = c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k}$ is the average number of one-degree columns remaining in the decoding process. For more information and numerical examples, see Chapter 50 in MacKay's book.

Implement a Matlab function, that, for given parameters k, n, c, and δ, calculates a random generator matrix G and the robust soliton distribution R. I assume the Matlab function in the form:

```
function [ G R ] = lt_generator_matrix( k, n, c, delta )
%LT_GENERATOR_MATRIX calculates k by n generator matrix of [n,k] LT code with
% parameters c and delta, i.e., columns are random binary vectors with the number
% of ones following the robust soliton distribution R with parameters c and delta.
end
```

Use the following parameters for testing your procedure: $c = 0.1$, $\delta = 0.5$, $k = 1000$, and $n = 2000$.

Problem 3: (experiments)

First, by using the function "peeling_solver.m", implement the decoding algorithms for LDPC codes (call it "ldpc_bec_decode.m") and the decoding algorithm for block LT codes (call it "lt_block_decode.m") over the BEC. These should be really short (one or two lines of code).

Now you have all the tools you need to communicate over the BEC and we will run some small experiments by using different LT and LDPC codes. In our experiments, we fix the code (represented either as H or G) to length $n = 2000$ in the beginning and test it over BEC's with different erasure probabilities. Use the following pseudo-code to run one experiment.

```
Create code C (construct either H for LDPC or G for LT code) of size 1000 x 2000
If C is an LDPC, then run the preprocessing step (ldpc_preprocess.m)
For every erasure probability e in [0:0.02:0.5]
  set s = random binary vector of 1000 information bits
  encode s into codeword x
  send x over the BEC with erasure probability e and obtain y (use bec_channel.m)
  run the decoding algorithm with input word y
  s_hat = extracted INFORMATION BITS // ( s_hat(i)=-1 if it is unknown )
  set P(e) = average number of unknown INFORMATION BITS = mean(s_hat==-1)
End for
Plot graph of P(e) versus e.
```

Run one experiment for each of the following codes (and put everything to file "experiments.m"):

1. (3,6) regular LDPC code $L(x) = x^3$, $R(x) = x^6$.
2. Irregular LDPC code with $L(x) = 0.4994x^2 + 0.3658x^3 + 0.0581x^7 + 0.0767x^{13}$, $R(x) = x^7$.
3. LT code with parameters $c = 0.1$, $\delta = 0.5$, $k = 1000$, and $n = 2000$
4. LT code with parameters $c = 0.03$, $\delta = 0.9$, $k = 1000$, and $n = 2000$.

What to submit as a solution:

As an output of this problem I want the following:

- Complete code listing of the functions "peeling_solver.m", "lt_generator_matrix.m", "experiments.m" and all other functions you use.
- One graph showing the performance comparison from all four experiments (4 curves).