Low-Density Parity-Check (LDPC) Codes
EECS 869: Error Control Coding
Fall 2013

Complete the following tasks. You should submit an e-mail with two .m file attachments (use the e-mail address esp@eecs.ku.edu).

1. **Implement the Iterative Log Likelihood Decoding Algorithm for Binary LDPC Codes.** This decoding algorithm is found on Moon p. 652 and is reproduced for your convenience on the following page (Moon also provides his own MATLAB implementation of the algorithm, which you must not consult as you implement your own). You should implement this algorithm as a MATLAB function with the following syntax:

   ```
   c_hat = LdpcLogDecXXX(r,A,NumIter);
   ```

   where it is understood that the input argument **r** has already been scaled by the channel reliability L_c, i.e. at runtime, when you call your function, you will pass in $L_c\mathbf{r}$. The input argument **A** is the low-density parity-check matrix and `NumIter` is the number of decoding iterations. As you can see from the input arguments and the algorithm specification, the implementation doesn't care whether **A** is large or small. However, it is easier to debug the implementation when **A** is small. In Example 15.7, Moon provides a numerical example, which you might find helpful, using the small LDPC code specified in Equation (15.1). I have provided this LDPC code for you (the **G** and **A** matrices both) in the file `Asmall.mat`. I have also provided a larger example in `Alarge.mat`. I will test your decoder implementation for both of these LDPC codes.

2. **Implement the "Scaled-Min" Approximation for the LDPC Decoding Algorithm.** This approximation uses a modified version of Equation (15.33) in Moon's algorithm specification:

$$\eta_{m,n}^{[l]} \approx -K\left(\prod_{j\in\mathcal{N}_{m,n}} \operatorname{sign}\left(-\lambda_j^{[l-1]} + \eta_{m,j}^{[l-1]}\right)\right) \min_{j\in\mathcal{N}_{m,n}} \left\{\left|-\lambda_j^{[l-1]} + \eta_{m,j}^{[l-1]}\right|\right\} \tag{1}$$

   where the scale factor K serves the purpose of preventing the approximate LLRs from growing too rapidly ($K = 0.75$ is a reasonable value). One way to implement (1) is to define a two-argument function, e.g.

$$f(a,b) = \operatorname{sign}(a)\operatorname{sign}(b)\min(|a|,|b|)$$

   [see Moon Equation (A.2)], and then call the two-argument function within a `for` loop to arrive at the "cumulative" result, just as a sum or product is computed within a `for` loop. Before starting your loop, you should initialize the result with the "identity" element, i.e. we initialize a sum with 0 and a product with 1, in this case the identity is given in Moon Equation (A.4).

   You should implement this version of the decoding algorithm as a MATLAB function with the following syntax:

   ```
   c_hat = LdpcScaledMinDecXXX(r,A,NumIter);
   ```

   In this case, the input argument **r** does *not* need to be scaled by the channel reliability L_c. Thus, while this version of the decoder is suboptimal by a few tenths of a dB, its metrics are simpler to compute and it does not require a SNR estimate. I will test your scaled-min decoder implementation for both of the LDPC codes mentioned previously.

Algorithm 15.2 Iterative Log Likelihood Decoding Algorithm for Binary LDPC Codes

Input: A, the received vector $\mathbf{r}$, the maximum # of iterations L, and the channel reliability L_c.
Initialization: Set $\eta_{m,n}^{[0]} = 0$ for all (m, n) with $A(m, n) = 1$.
Set $\lambda_n^{[0]} = L_c r_n$
Set the loop counter $l = 1$.
Check node update: For each (m, n) with $A(m, n) = 1$: Compute

$$\eta_{m,n}^{[l]} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda_j^{[l-1]} - \eta_{m,j}^{[l-1]}}{2}\right)\right) \tag{15.33}$$

Bit node update: For $n = 1, 2, \ldots, N$: Compute

$$\lambda_n^{[l]} = L_c r_n + \sum_{m\in\mathcal{M}_n} \eta_{m,n}^{[l]} \tag{15.34}$$

Make a tentative decision: Set $\hat{c}_n = 1$ if $\lambda_n[l] > 0$, else set $\hat{c}_n = 0$.
If $A\hat{\mathbf{c}} = 0$, then **Stop**. Otherwise, if #iterations $< L$, loop to **Check node update**
Otherwise, declare a decoding failure and **Stop**.