Design and Implementation of Low Density Parity Check Codes

Prasanna Sethuraman
Munich University of Technology
prasanna@mytum.de

Abstract

This article explains the design and implementation of both regular and irregular LDPC codes as carried out by me at the Institute of Communications Engineering, Technical University of Munich. The design of a good irregular LDPC code needs a pair of distributions to be specified. This article discusses a randomized construction which attempts to form a parity check matrix whose row and coloumn densities are as close as possible to the distributions specified. The encoding and decoding of the designed LDPC codes are also discussed.

1 Introduction

It has been known recently that LDPC codes can outperform the best known Turbo codes if designed properly. Richardson *et al*.[1] has shown that construction of good LDPC code requires an irrgular distribution of check node and variable node degrees of the parity check matrix. Further, a good irregular degree distribution can be obtained with density eveolution. It then remains to explicitly specify a method of constructing LDPC codes with the specified number of ones in each row and each coloumn. We should also take care to eliminate cycles of length four in the LDPC code. I implemented a randomized construction of such a parity check matrix using the bit filling approach.

The works that look at methods of construction are by David J.C. MacKay[2, 3] and Campello *et al*[4]. MacKay discusses randomized constructions of quasi-regular LDPC codes. MacKay's constructions acheive regular column weights but not regular row weights. The construction by Campello is more interesting since it tries to construct graphs of large girth. Their construction can also handle irregular column distributions. However, the algorithm does not explicitly include a way to also satisfy the row distribution constraint. Quite recently a lot of work is being done of construction of parity check matrices with specified girth. In this work, we constructed codes that contain no length four cycles (girth of atleast 6) and satisfy both the specified degree distribution constraints. The organization of this document is as follows. In section 2 we describe the algorithm for code construction. In section 3 we describe a few straightforward encoding procedures. In section 4 we describe the decoding procedure that we followed. All routines were implemented in matlab, and surprisingly, we were able to simulate P_e of 10^{-5} within a few hours. Section 5 presents the simulation results and section 6 concludes.

2 Code Construction

The objective is to design a parity check matrix of size $M \times N$ with number of ones in rows and coloumns distributed according to the specified degree distribution pair. In this article, we specify the check node degree distribution as *dc* and *dv* which are $2 \times K$ arrays. The first row contains number of ones for a fraction of rows (or coloumns) and the second row contains the fraction of rows (or coloumns). The algorithm that we used is described next.

We start with an empty matrix, and for each coloumn, select a random row and assign ones to that (row, coloumn) till the number of ones in that coloumn equals the required number for that coloumn. Before assigning a one, the following checks are made:

- Check if the degree constraint for the corresponding row is violated.
- Check if any cycles of length four will be formed.

If any of the above two conditions are violated, select and another random row and check again. Continue till a row is found which together with that coloumn satisfies the above two constraints. Note that for the algorithm to be able to distribute ones properly the degree distribution equation

$$M = N\frac{\sum d_v}{\sum d_c}$$

should be satisfied.

3 Encoding

Encoding uses the fact that any generator matrix (and equivalently the parity check matrix) for a linear block code can be represented in a systematic form. The encoder output is then $x = [u\ c]$ where u is the information bit sequence and c is the parity check bit sequence. There are various ways of forming the code word x. One way is to construct the generator matrix explicitly by reducing the parity check matrix to a *reduced row echelon form*. This requires the use of Gaussian Elimination which is a costly operation when dealing with matrices of large size. The number of elementary operations required for Gaussian elimination is $O(n^3)$ where n is one of the matrix dimensions[12]. The parity check matrix converted to the reduced row echelon form is of the form $H = [I\ P^T]$, dimensions of H being $M \times N$ and P being $M \times (N-M)$. Generator matrix can then be readily formed as $G = [P^T\ I]$, dimension of G being $(N-M) \times N$.

Another method is to use directly the parity check matrix. Let $H = [A\ B]$. Now,

$$Hx' = [A\ B]\begin{bmatrix} u' \\ c' \end{bmatrix} \tag{1}$$

$$= Au' + Bc' \tag{2}$$

$$= 0 \tag{3}$$

because Hx = 0 if x is a codeword. Thus,

$$c' = B^{-1}Au' \tag{4}$$

For the above procedure to work, we must make sure that an inverse exists for B. [5] presents an algorithm similar to Gaussian Elimination to decompose B into LU form (I tried it, but it did not work for me, nor did Gaussian Elimination). In our simulation, we transmitted the all zero code word, since doing the Gaussian elimination for block lengths of interest seemed next to impossible in matlab.

4 Decoding

There are various methods of representing the bipartie graph required for decoding. One way is to look at the bipartite graph as an "interleaver". The decoder structure exactly resembles the decoder of a Repeat Accumulate code, only without the accumulator[6]. The received vector is passed on to the Variable Node Decoder (VND) which in the first round simply repeats the elements of the received vector. The output of the variable node decoder is then passed into the interleaver the output of which is fed in to the Check Node Decoder (CND). The CND's output is then deinterleaved and passed to the VND.

It only remains now to explicitly specify the interleaver, given the parity check matrix. There are two ways. One is to use a block interleaver, like the classical approach. Write into the parity check matrix row wise and read from it column wise. And vice versa for the deinterleaver. However, there is another better way. It is best made clear with an example.

Let the H matrix be as shown (for example)

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

The values received from the channel are passed to a Variable Node Decoder, whose output is

$$VND\ output: [1\ 1\ 2\ 2\ 3\ 3\ 4\ 5\ 5\ 6\ 6\ 7\ 7\ 7]$$

The numbers correspond to the variable nodes. The output of VND is then rearranged (interleaved) and passed to the Check Node Decoder, whose output is:

CND output:	[1	4	6	2	3	7	1	2	5	7	3	5	6	7]
Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Thus the interleaver is just

$$Interleaver = [1,7,4,8,5,11,2,9,12,3,13,6,10,14]$$

The implementation of CND and VND uses the forward-backward recursion described in [7].

5 Simulation Results

The performance of a (N=2000, K=1000) LDPC code on an AWGN channel is shown in figure together with the performance of uncoded BPSK. The LDPC code was generated by the random construction method described above and does not have any cycles of length four. Several facts about iterative decoding are verified with the curves shown(also see [8]).

1. For low $\frac{E_b}{N_0}$ the decrease in P_e is negligible with increase in number of iterations.
2. For moderate values of $\frac{E_b}{N_0}$, (0,1 and 2 dB), P_e keeps on decreasing as the number of iterations increase (well... kind of).
3. For high $\frac{E_b}{N_0}$, ($\geq$ 3 dB), P_e decreases rapidly in first few iterations (10-15 iterations typically). Beyond that, the gain acheivable with increasing number of iterations is minimal (also confirmed in [9]).

6 Conclusion and Future Work

The algorithm for constructing LDPC codes using the Extended Bit-Filling algorithm is certainly attractive. A method of performing encoding iteratively has been studied by Grant[10]. This greatly reduces the encoding complexity. I am also working on analysis of LDPC code constructions using finite geometries. Finite geometry constructions are of interest because they provide a large minimum distance and the code construction is extremely simple. A performance comparison of all known constructions of LDPC codes is the target. Such a performance comparison will be very useful in giving insights into the trade-offs between different parameters that influence the performance of LDPC codes.

References

[1] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes", IEEE Transactions on Information Theory, Vol. 47, No. 2, February 2001.

[2] D. J. C. MacKay, and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes", Downloaded from WWW.

[3] D. J. C. MacKay, http://wol.ra.phy.cam.ac.uk/mackay.

[4] J. Campello, and D. S. Modha, "Extended Bit-Filling and LDPC Code Design", Proc. of the IEEE Globecom Conference, Nov. 25-29, 2001.

[5] Radford M. Neal, `www.cs.utoronto.ca/~radford/`

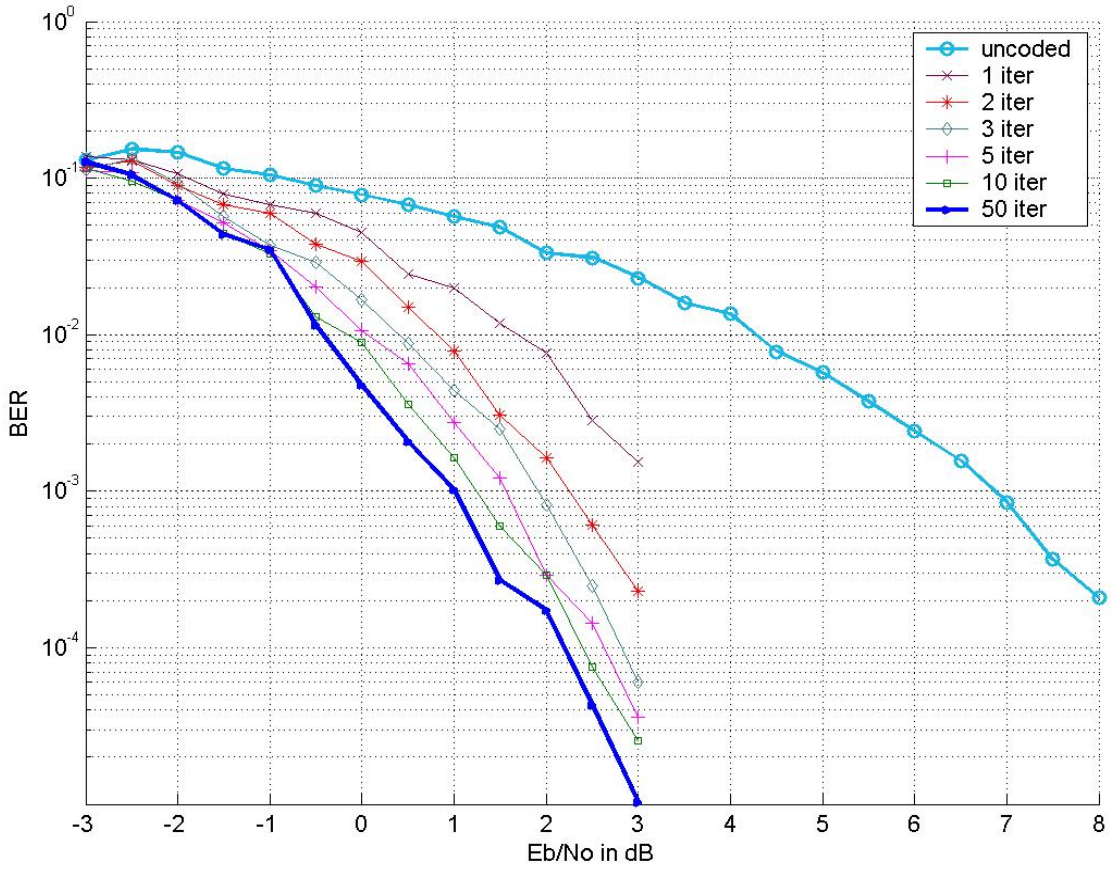

Figure 1: Performance of N=2000, K=1000 regular (3,6) LDPC code

[6] S. ten Brink and G. Kramer, "Design of Repeat-Accumulate Codes for Iterative Detection and Decoding", submitted to IEEE special issue on Signal Processing for MIMO, Dec. 2002.

[7] Xiao-Yu Hu, E. Eleftheriou, D. M. Arnold, and A. Dholakia, "Efficient Implementations of the Sum -Product Algorithm for Decoding LDPC Codes", Proc. 2001 IEEE GlobeCom Conf., Nov. 2001.

[8] S. ten Brink, "Convergence Behaviour of Iteratively Decoded Parallel Concatenated Codes", IEEE Transactions on Communications, V0l 49, No. 10, October 2001.

[9] D. Waters, `http://users.ece.gatech.edu/~deric/Projects/ldpc/LDPC_code.htm`

[10] D. Haley, A. Grant, and J. Buetefuer, "Iterative Encoding of Low-Density Parity-Check Codes", Downloaded from WWW.

[11] J. Rosenthal, P. O. Vontobel, "Constructions of LDPC Codes using Ramanujan Graphs and Ideas from Margulis", Proc. of 38^{th} Allerton Conference on Communication, Control and Computing, Illinois, October 4-6, 2000.

[12] G. Strang, "Linear Algebra and its Applications", Chapter 1, page 15.