LDPC Codes, Construction and Applications

Manu T S

October 20, 2013

Abstract

In this report we study construction of LDPC codes and implementation of encoder and decoder for LDPC codes. We implement the progressive edge growth algorithm, algorith for construction of LDPC codes based on Reed-Solomon Codes. We also implement an encoder for LDPC codes. For decoder belief propogation decoder is implemented.

1 Motivation

Coding Theory is one of the foundations of Communication Engineering. Search for better and better codes has always been an area of intense research. For a long time there was a significant gap between the theoretical capacity and practically achievable capacity. It is proved that LDPC codes introduced by Gallager, form a class of capacity approaching codes. The research in that area was dormant for a long time till they were rediscovered in the late 1990s[6]. It is shown that these codes decoded with iterative decoding algorithms like sum-product algorithm[6] or message passing algorithms[1] achieve good error performance.

GNU Radio provides a free and open-source platform for implementing practical communication systems using various compatible hardware like USRP, and also provides a learning and simulation tool. The use of GNU Radio in research related to Software Defined Radio, Digital Signal Processing and Communication Engineering is becoming increasingly widespread. In this project we implement a generic encoder and decoder for LDPC codes in GNU Radio and also implement algorithms for construction of LDPC codes.

2 LDPC Codes

The LDPC codes, as the name suggests is characterized by a sparse parity check matrix $\mathbf{H}$. From parity check matrix we can construct the *Tanner graph* of the code. For example fig. 1 represents the Tanner graph of [7, 4, 3] Hamming code with

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

A codeword $\mathbf{x}$ from the codebook described by parity check matrix $\mathbf{H}$ should satisfy

$$\mathbf{Hx} = \mathbf{0} \tag{2}$$

In the Tanner graph of a (t_c, t_r) regular binary LDPC code, each variable node will have degree t_c and each check node will have degree t_r. Irregular LDPC codes can have different degrees for nodes. Let λ and ρ be vectors such that their i^{th} component λ_i and ρ_i represent the fraction of edges connecting to a variable node of degree i and check node of degree i respectively. Thus

$$\lambda(x) = \sum_i \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) = \sum_i \rho_i x^{i-1}$$

are called variable and check degree distribution polynomials respectively. Standard ensemble LDPC(n, λ, ρ) of bipartite graphs is defined as the set of bipartite graphs with n variable nodes, variable degree distribution $\lambda(x)$ and check degree distribution $\rho(x)$.

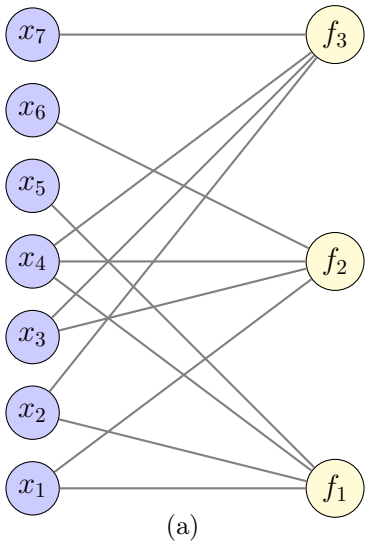


Figure 1: Tanner graph of [7, 4, 3] Hamming code with **H** given in eq. (1). The blue nodes on the left represents the components of a codeword. They are called variable nodes. The yellow nodes on the right represents each of the constraints in the system eq. (2). They are called check nodes. When $h_{i,j}$ is 1 there exists a link between j^{th} variable node and i^{th} check node.

3 Marginalization Via Message Passing

Consider a function f with factorization:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5) \tag{3}$$

Now, the marginal of function f w.r.t x_1 is:

$$f(x_1) = \sum_{x_2, x_3, x_4, x_5, x_6} f(x_1, x_2, x_3, x_4, x_5, x_6) \tag{4}$$

$$= \left[\sum_{x_2, x_3} f_1(x_1, x_2, x_3)\right] \left[\sum_{x_4} f_3(x_4) \left(\sum_{x_6} f_2(x_1, x_4, x_6)\right) \left(\sum_{x_5} f_4(x_4, x_5)\right)\right] \tag{5}$$

Factor graph associated with above factorization is given in Figure 2. Here the red circles represent the variables $(x_1, x_2, \cdots, x_6)$ and are called the *variable-nodes*. Blue circles represent the factor (f_1,f_2,f_3 and f_4) and referred as the *check-nodes*. In the factor graph, a variable-node is connected to a check-node iff the corresponding variable takes part in that factor. Notice that

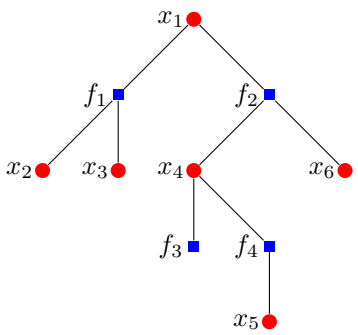


Figure 2: Factor Graph

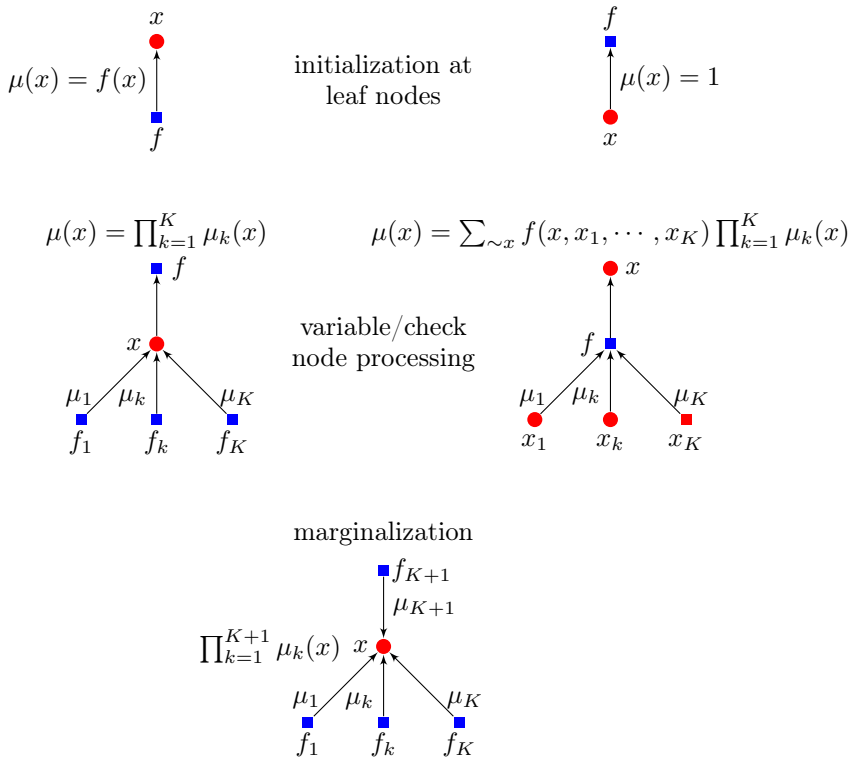


Figure 3: Message Passing Rules

this factor graph is a tree (i.e. there is no cycle in the graph). In the case where the factor graph is a tree the marginalization problem can be broken into smaller tasks according to the structure of the tree. The algorithm is explained in the next paragraph

The alogorithm proceeds by sending messages along the edges of the tree. Messages are functions on $\mathcal{X}$, or equivalently, vectors of length $|\mathcal{X}|$. The messages signify marginals of parts of the function and these parts are combined to form the marginal of the whole function. Message passing originates at the leaf nodes. Messages are passed up the tree and as soon as a node has received messages from all its children, the incoming messages are processed and the result is passed up to the parent node. Figure 3 explains the computation to be performed at the nodes during marginalization.

To compute tha marginal with respect to all the variables, one can consider for each variable, a tree rooted in this variable and execute single marginal message passing algorithm on each rooted tree. Here all the computations can be performed simultaneously on a single tree. Start at all leaf nodes and for every edge compute the outgoing message along this edge as soon as you have the incoming messages along all other edges that connect to a given node.

4 Decoding Via Message Passing

For memoryless channel without feedback, bitwise maximum a posteriori (MAP) decoding of a codeword is given by [1]:

$$\hat{x}_i(y) = \arg\max_{x_i} p_{X_i|Y}(x_i|y) \tag{6}$$

$$= \arg\max_{x_i} \sum_{\sim x_i} p_{X|Y}(x|y) \tag{7}$$

$$= \arg\max_{x_i} \sum_{\sim x_i} p_{Y|X}(y|x) p_X(x) \tag{8}$$

$$= \arg\max_{x_i} \sum_{\sim x_i} \prod_j p_{Y_j|X_j}(y_j|x_j) p_X(x) \tag{9}$$

The notation $\sum_{\sim x_i}$ indicates that sum is over all components of x, except x_i. We obtain (7) from (6) by total probability rule. On application of Bayes's rule, we obtain (8) from (7). Equation (7) has very high computational complexity. For a codeword of length n and rate R, approximate computational complexity is $\mathrm{O}(2^{nR})$. The computational complexity can be reduced by factorizing the sequence probability. To this end, in equation(9), we use the fact that the channel is memoryless, without feedback. Furthermore, since all the input codewords are equiprobable,

$$\hat{x}_i(y) = \arg\max_{x_i} \sum_{\sim x_i} \prod_j p_{Y_j|X_j}(y_j|x_j) \mathbf{1}_{\{x \in C\}} \tag{10}$$

Where $\mathbf{1}_{\{x \in C\}}$ is a code membership function for the codebook C. Code membership function $\mathbf{1}_{\{x \in C\}}$ has a factorized form, and this factorization is used in efficient decoding.

4.1 Message Passing decoder for Binary Erasure Channel

By the standard message-passing rules the initial messages are $(\mu_j(0), \mu_j(1)) = (p_{Y_j|X_j}(y_j|0), (p_{Y_j|X_j}(y_j|1))$. In the case of binary erasure channel, the messages are either $(1-\epsilon, 0), (\epsilon, \epsilon)$, or $(0, 1-\epsilon)$ corresponding to 0, ? (erasure), or 1 respectively received from channel output. In this case we can normalize the message to (1, 0), (1, 1) and (0, 1) correspondingly. Now the message processing rules are discussed.

Claim : For BEC general message passing rules shown in Figure 3 simplify to the following. At a variable node the outgoing message is an erasure if all incoming messages are erasures. Otherwise, since the channel never introduces errors, all non-erasure messages must agree and either be 0 or 1. In this case the outgoing message is equal to this common value. At a check node the outgoing message is an erasure if anu of the incoming message is an erasure. Otherwise, if all the incoming messages are either 0 or 1, then the outgoing message is the mod-2 sum of the incoming messages

Proof : If all messages entering a variable node are from the set $\{(1,0),(1,1)\}$, then the outgoing message, which is equal to the component-wise product of the incoming messages according to the general message-passing rules, is also from this set. Further, it is equal to (1, 1) only if all incoming messages are of the form (1, 1) i.e, erasures. The equivalent statement is true if all incoming messages are from the set $\{(1,0),(1,1)\}$. Since channel never introduces errors, we only need to consider these two cases.

Next we consider the claim concerning the message-passing rule at a check node: We consider only a check node of degree 3 with two incoming messages, since check node of higher degree can be modeled as the cascade of several check nodes, each of which has 2 inputs. Let $(\mu_1(0), \mu_1(1))$, and $(\mu_2(0), \mu_2(1))$ denote the incoming messages. By the message processing rules the outgoing

message is

$$(\mu(0),\mu(1)) = \left(\sum_{x_1,x_2} \mathbf{1}_{\{x_1+x_2=0\}}\mu_1(x_1)\mu_2(x_2), \sum_{x_1,x_2} \mathbf{1}_{\{x_1+x_2=1\}}\mu_1(x_1)\mu_2(x_2)\right) \quad (11)$$

$$= (\mu_1(0)\mu_2(0)+\mu_1(1)\mu_2(1), \mu_1(0)\mu_2(1)+\mu_1(1)\mu_2(0)) \quad (12)$$

If $(\mu_j(0),\mu_j(1)) = (1,1), j \in \{1,2\}$, then after normalization, $(\mu(0),\mu(1)) = (1,1)$. Thus if any of the input is an erasure then output is an erasure. Also if both messages are known, then by explicite check, we can see that the output corresponds to mod-2 sum. Figure 4 shows an example of this decoder on (7, 4, 3) Hamming code with parity check matrix given in Equation(1)

4.2 Message Passing Decoder for Binary Input AWGN Channel

Given in Figure 5 is a mathematical model of AWGN channel. We think that the channel input $x_i \in \{-1,+1\}$. Here again a message can be thought of as a real-valued vector of length 2, $(\mu(-1),\mu(1))$. We initialize the message passing iterations by sending from i^{th} variable node the vector $(p_{Y_i|X_i}(y_i|-1), p_{Y_i|X_i}(y_i|1))$. In case of binary input channels, it suffices to send the log-likelihood ratios.

$$l = \log\left(\frac{\mu(-1)}{\mu(1)}\right)$$

On using log-likelihood ratios, the message passing rules simplipy to the following. At variable node, simply add the incoming messages and at a check node of degree $J+1$, use the following computation.

$$l = 2\,\mathrm{arc\,tanh}\left(\prod_{j=1}^{J}\tanh\left(l_j/2\right)\right) \quad (13)$$

In the case of Binary Input AWGN Channel, we can initialize the message passing iterations by sending

$$l_i = \log\left(\frac{\mu(-1)}{\mu(1)}\right) \quad (14)$$

$$= \log\left(\frac{p_{Y_i|X_i}(y_i|-1)}{p_{Y_i|X_i}(y_i|1)}\right) \quad (15)$$

$$= \frac{-2y_i}{\sigma^2} \quad (16)$$

from variable nodes to the check nodes. Further computations are carried out as expained above. One iteration consists of message passing from variable nodes to check nodes, computations at the check nodes and message passing from check nodes to variable nodes. After one such iteration, we find the probability ratios by the marginalization step shown in the Figure 3. Based on the probability ratios, we estimate the codeword, by the following rule. If the probability ratio is greater than 0, we decode that bit in favor of -1 and else in favor of 1. We check if the estimated codeword is an actual codeword by parity checks. If the estimated codeword is an actual codeword, we stop, else we continue with the next round of message passing iterations.

5 Construction of the parity check matrix using Reed-Solomon code

The parity check matrix of a LDPC code is a sparse matrix. Ideally the Bipartite graph constructed using the parity check matrix should not contain a cycle (no feedback) for optimal decoding. However, in practice it is hard to generate parity check matrices without cycles. Nevertheless most well defined methods for generating LDPC codes try to avoid cycles of length less

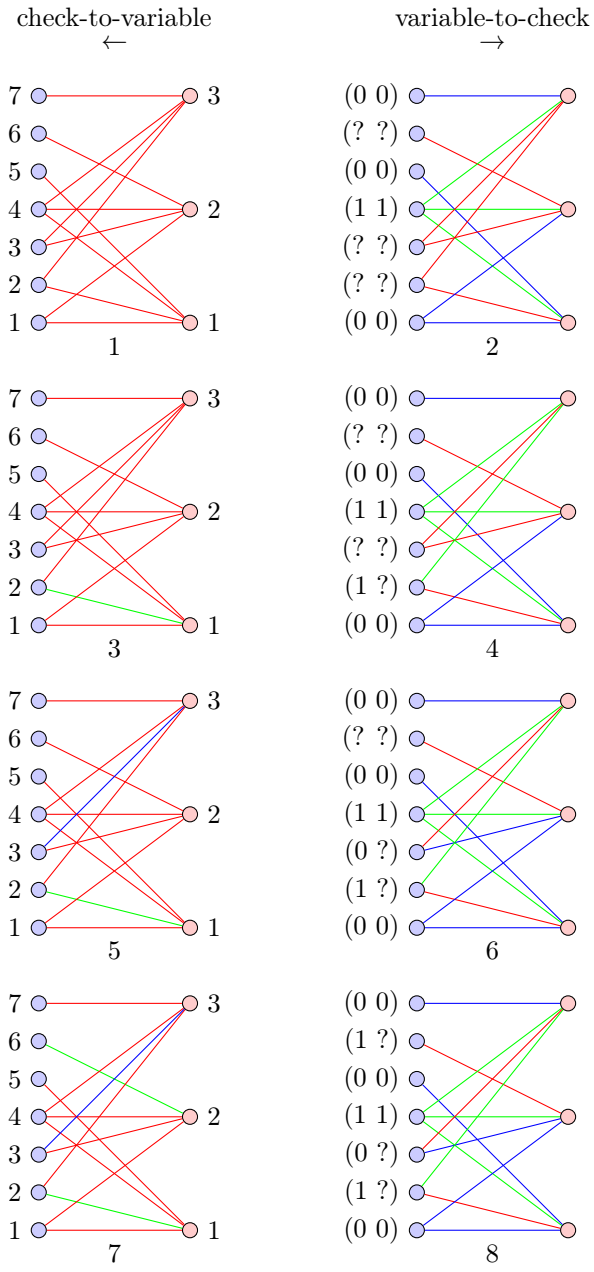


Figure 4: Message-passing decoding of the [7, 4, 3] Hamming code with the received code word y = (0, ?, ?, 1, 0, ?, 0). The left column is represents message passing from check to variable nodes. Right column represents message passing from variable to check nodes. Red links represents erasure messages, blue links represents messages 1 (corresponds to codebit 0) and green links represents message -1 (corresponds to codebit 1). The labels on the variable nodes in the right column represents the pair (estimated bit, received bit)

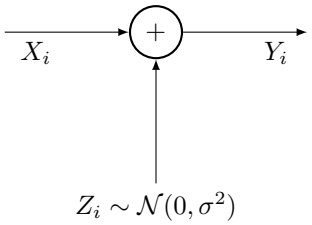


Figure 5: AWGN Channel

than some minimum length (minimum length could be any of the even positive integer like 6, 8, 10, etc.). In this chapter we will discuss LDPC code construction based on Reed-Solomon codes[**?**], also known as RS-LDPC codes. The bipartite graph for these codes does not have cycle of length 4. A $(6, 32)$ regular $(2048, 1723)$ RS-LDPC code has been adopted as the FEC in the IEEE 802.3an 10GBase-T standard[**?**].

5.1 Reed-Solomon codes

Consider the Galois field $GF(q)$ with q elements, where q is a positive integer power of a prime number. Let ρ be a positive integer such that $2 \leq \rho < q$. The generator polynomial of cyclic (n, k, d_{min}) code C is given by:

$$g(X) = (X - \alpha)(X - \alpha^2) \cdots (X - \alpha^{\rho-2}) \tag{17}$$

$$= g_0 + g_1 X + \cdots + X^{\rho-2} \tag{18}$$

Notice that $n = q - 1$, $k = q - \rho + 1$, $g_i \in GF(q)$ and α is a primitive element of a field. The parity check matrix R_H for a Reed-Solomon code has size $(\rho - 2) \times n$. Any submatrix of size $(\rho-2)\times(\rho-2)$ of parity check matrix is the Vandermonde matrix. Therefore linear combination of any $(\rho - 2)$ column will not result in 0_v. The rank of matrix R_H can be utmost $(\rho - 2)$. Thus minimum distance is $d_{min} = (\rho - 1)$.

Now consider the $(q-1)$-tuple vector $g^{(0)} = (g_0, g_1, ..., g_{\rho-2}, 0, 0, ..., 0)$. Notice that $g_{\rho-2} = 1$. By cyclically shifting $g^{(0)}$, we get generator matrix G of size $k \times n$ for code C.

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & . & . & 1 & 0 & . & . & 0 \\ 0 & g_0 & g_1 & g_2 & . & . & 1 & .0 & . & . \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & . & . & g_0 & g_1 & g_2 & . & . & . & 1 \end{pmatrix}$$

The generator matrix for shortened RS code C_b is a submatrix of size $2 \times \rho$ and it is shown below:

$$G_b = \begin{pmatrix} g_0 & g_1 & g_2 & . & . & . & 1 & 0 \\ 0 & g_0 & g_1 & g_2 & . & . & . & 1 \end{pmatrix}$$

5.2 Properties of C_b

1. Since the length of codewords in C_b is ρ and the minimum distance between two codewords of C_b is $(\rho - 1)$, two codewords in C_b only agree at most at one location.

2. Let c be a codeword of weight ρ. If we multiply c by $\forall \beta \in GF(q)$, we get set $C_b^{(1)}$ of $(q-1)$ codewords of weight ρ. Now, length of every codeword of $C_b^{(1)}$ is also ρ. So $C_b^{(1)}$ is a MDS (Maximum Distance Separable) code.

3. Let us partition C_b into a q cosets $C_b^{(1)}, C_b^{(2)}, ..., C_b^{(q)}$ based on $C_b^{(1)}$. Notice that $C_b^{(i)}$ is MDS code. Therefore two codewords in any coset $C_b^{(i)}$ must differ in all the locations.

5.3 Construction

Let us now explain the explicit construction procedure for a LDPC code check matrix H.

1. All q elements of $GF(q)$ can be expressed as some power of a primitive element α. Let us define the location vector of α^i as a q-tuple over $GF(2)$,

   $$z(\alpha^i) = (0, 0, ..., 1, 0, ..., 0)$$

   , where i^{th} element of $z(\alpha^i)$ is 1 and all other elements are 0. Choose one codeword $b = (b_1, b_2, ..., b_\rho) \in C_b^{(i)}$. If we replace each b_i $(1 \leq i \leq \rho)$ by its location vector $z(b_i)$, we get a $z(b) = (z(b_1), z(b_2), ..., z(b_\rho))$, which is a ρq-tuple of weight ρ over $GF(2)$.

2. Arrange all q ρq-tuple of $C_b^{(i)}$ as rows of a matrix and call this matrix as A_i. The weight of each column of A_i is 1.

3. Choose a positive integer γ, such that $1 \leq \gamma \leq q$. Then the parity check matrix H of size $\gamma q \times \rho q$ is defined as:

   $$H \triangleq \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_\gamma \end{bmatrix} \tag{19}$$

Since each column of A_i has weight 1, weight of an each column of H is γ. So, H is a (γ, ρ)-regular matrix. Each row in A_i is a coset member, so each row in A_i is different,

1. Two rows in A_i do not have single common element,

2. Two codewords in C_b agree at at most one symbol location ($d_{min} = \rho - 1$).

From Property 1 and 2 above, no two rows from A_i, A_j, $i \neq j$ agree at more than a single element. This will imply that the bipartite graph corresponding to H is free of length 4 cycles. To see this, notice that our construction guarantees a H which contains no rectangle having all the corner points non-zero.

In this chapter we have shown the construction of a (γ, ρ) regular RS-LDPC. This code has girth at least six. In the next chapter we show the construction of an irregular LDPC code.

6 Construction of Parity Check Matrix using Progressive Edge Growth

In this section, we present progressive edge growth (PEG) method for the construction of regular and irregular LDPC code. A bipartite graph can be described using bit nodes, check nodes and set of edges E. In this method, edges between bit nodes and check nodes are established in a progressive manner. For given bit node, an edge connects the one of the check node such that girth is maximum. Thus, PEG algorithm has large girth when compared to codes constructed using random methods. Hence, code constructed using PEG algorithm has low error floor in comparison with code constructed using random methods.

Let us describe bit node degree sequence as follow:

$$D_b = \{d_{b1}, d_{b2}, \cdots, d_{bn}\},\ d_{b1} \leq d_{b2} \leq \cdots \leq d_{bn}$$

Where d_{bi} presents the degree of i^{th} bit node.

Algorithm 1 Progressive Edge Growth Algorithm[4]

Input: sequence D_b
Output: parity check matrix H
initialize all check nodes with degree 0
for $i = 1$ to n **do**
  for $j = 1$ to d_{bi} **do**
    if $j = 1$ **then**
      1. find minimum degree check nodes set $C = \{c_1, c_2, \cdots, c_m\},\ 1 \leq m \leq n-k$ such that $deg(c_1) = deg(c_2) = \cdots = deg(c_m)$
      2. choose check node $c_1 \in C$
      3. put an edge between i^{th} bit node and check node c_1
      4. increase the degree of c_1 by 1
    else
      1. for i^{th} bit node find check nodes set $C = \{c_1, c_2, \cdots, c_m\},\ 1 \leq m \leq n-k$ such that girth is maximum and $deg(c_1) \leq deg(c_2) \leq \cdots \leq deg(c_m)$
      2. put an edge between i^{th} bit node and check node c_1
      3. increase the degree of c_1 by 1
    end if
  end for
end for

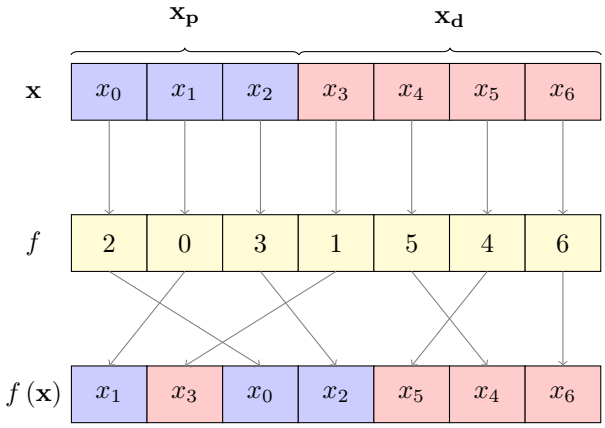
Check node degree distribution obtained using Algorithm 1 is almost uniform. Whenever multiple choices are available to pick check node from set C, we can either pick first check node in the set C or pick any check node from set C. In algorithm 1, we always choose the first member of set C.

7 Encoding

If the parity check matrix is given as $[\mathbf{I}\ \mathbf{P}]$, where $\mathbf{I}$ is $N-K$ x $N-K$ identity matrix, $\mathbf{P}$ is some $N-K$ x K matrix, a codeword $\mathbf{x} = \left[\mathbf{x_p^T}\ \mathbf{x_d^T}\right]^\mathsf{T}$ can be formed by assigning

$$\mathbf{x_p} = \mathbf{P} \cdot \mathbf{x_d}$$

We obtain $\mathbf{G} = [\mathbf{I}\ \mathbf{P}]$ from $\mathbf{H}$ by Column permutation, Row additions and Row permutations Suppose f represents the permutation in obtaining $\mathbf{G}$ from $\mathbf{H}$, and $\mathbf{G} \cdot \mathbf{x} = \mathbf{0}$, then $\mathbf{H} \cdot f(\mathbf{x}) = \mathbf{0}$



Algorithm 2 Construction of $\mathbf{G} = [\mathbf{I}\ \mathbf{P}]$

Input: Parity Check matrix $\mathbf{H}$
Output: $\mathbf{G} = [\mathbf{I}\ \mathbf{P}]$
$nchk \leftarrow$ Number of rows of $\mathbf{H}$
$nvar \leftarrow$ Number of columns of $\mathbf{H}$
$limit \leftarrow$ Number of rows of $\mathbf{H}$
$rank \leftarrow 0$
$i \leftarrow 0$
$f \leftarrow (1, 2, 3, \ldots, nvar)$
while $i < limit$ **do** ▷ Pivoting at (i, i)
  $found \leftarrow$ False ▷ Flag indicating if the row contains non-zero entry
  for j from i to $nvar$ **do**
    if $\mathbf{H}[i, j] = 1$ **then** ▷ Encountering a non-zero entry at (i, j)
      $found \leftarrow$ True
      $rank \leftarrow rank + 1$ ▷ Increment rank
      swap columns i and j ▷ Now the entry at (i, i) is 1
      swap $f(i)$ and $f(j)$ ▷ Keeping track of the column permutation
      break
    end if
  end for
  if $found =$ true **then**
    for k from i to $nchk$ **do**
      if $\mathbf{H}[k, i] = 1$ **then** ▷ Checking for non-zero entries below (i, i)
        Add row i^{th} row to k^{th} row
      end if
    end for
  end if
  if $found =$ false **then** ▷ We encounter a row with all zeros
    for $rows$ in i to $limit$ **do**
      swap row $rows$ with $rows + 1$ ▷ Pushing the all-zero row to the bottom
    end for
    $limit \leftarrow limit - 1$
  end if
end while

8 Outcomes and Results

Following classes and blocks were desingned as part of this project. These can be installed as an Out of Tree module in GNU Radio. The source can be downloaded from [10]

- Class for GF(2) Vector arithmetic.
- Class for GF(2) Matrix arithmetic.
- Class for accessing LDPC codes in "alist-file"[9] format.
- Class for holding an LDPC code.
- Class for message passing mechanisms for BSC and AWGN channels.
- LDPC Encoder block.
- LDPC Decoder blocks, for both float inputs and byte inputs.
- Two algorithms for construction of LDPC codes

Table 1: Bit Error Rate Comparison

sigma	Errors (No code)	BER (No Code)	Errors (With LDPC)	BER (With LDPC)
0.3	4338	0.0004338	0	0
0.4	61814	0.0061814	0	0
0.5	227228	0.0227228	31	3.1e-06
0.6	477744	0.0477744	65855	0.0065855
0.7	765326	0.0765326	572393	0.0572393
0.8	1055848	0.1055848	968856	0.0968856
0.9	1332039	0.1332039	1291208	0.1291208
1.0	1585653	0.1585653	1565803	0.1565803

The performance of a (3, 6) regular LDPC code with $N = 96, K = 50$, and $M = 48$

Table 2: Block Error Rate Performance

sigma	Errors (No Code)	BER (No Code)	Errors (With LDPC)	BER (With LDPC)
0.3	3898	0.196461	0	0
0.4	18938	0.954488	0	0
0.5	19840	0.999994	0	0
0.6	19841	1.0	17	0.0008568
0.7	19841	1.0	19841	1.0

The performance of an LDPC code generated using PEG with $N = 1008, K = 504$, and $M = 504$

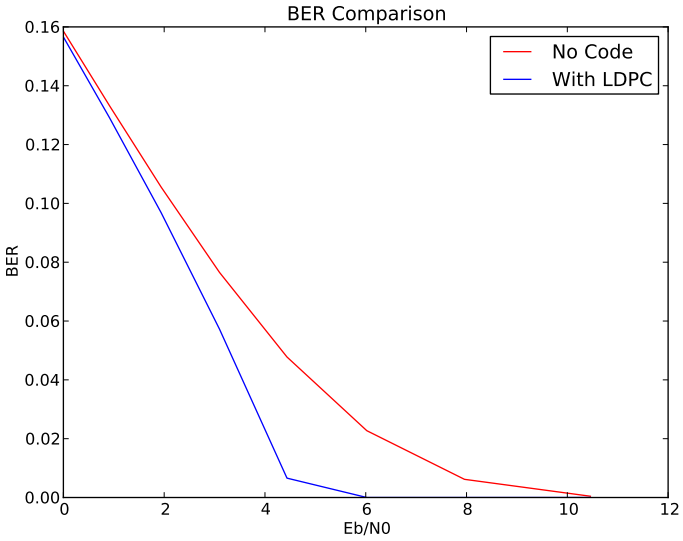


Figure 6: BER plot corresponging to Table 1

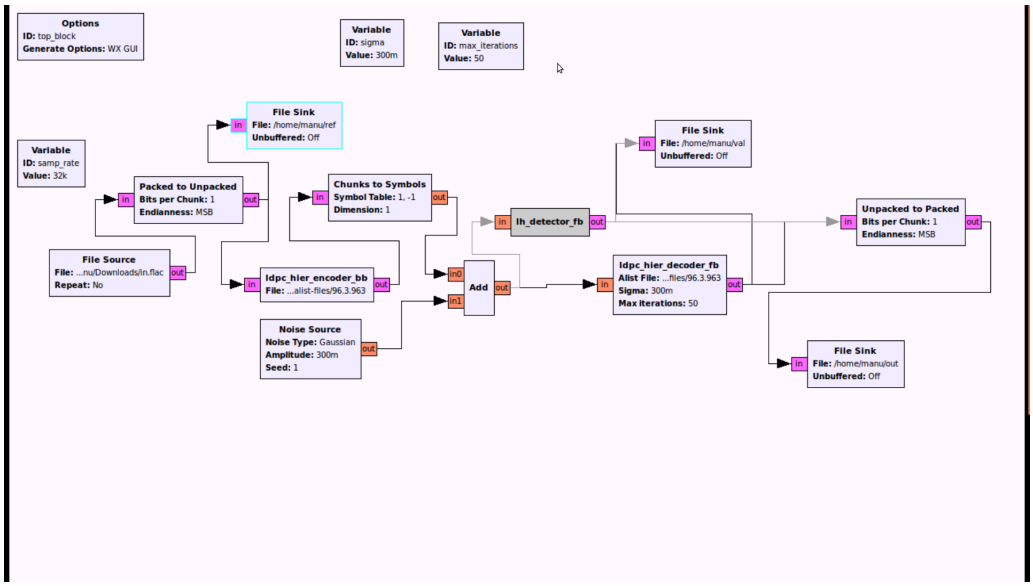


Figure 7: GNU Radio block diagram

Tables 1 and 2 compares the performance of these blocks with uncoded BPSK. The plot of Bit error performance is in Figure 6 The GNU Radio set up used in this experiment is shown in Figure 7

References

[1] T. Richardson and R. Urbanke, "Modern Coding Theory", Cambridge 2003.

[2] I. Djurdjevic, Jun Xu, K. Abdel-Ghaffar and Shu Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols", IEEE Communication Letters, July 2003.

[3] T. Richardson, M Amin Shokrollahi and R. Urbanke,"Design of Capacity Approaching Irregular Low-Density Parity-Check Codes", Transactions On Information Theory, February 2001.

[4] Xiao-Yu Hu, Evangelos Eleftheriou and Dieter M. Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graphs", Transactions on Information Theory, January 2005.

[5] Jun Xu, Lei Chen, I. Djurdjevic, Shu Lin and K. Abdel-Ghaffar, "Construction of Regular and Irregular LDPC codes: Geometry Decomposition and Masking", Transactions on Information Theory, January 2007.

[6] David J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices", Transactions on Information Theory, March 1999.

[7] Richard E. Blahut, "Algebraic Codes for Data Transmission", Cambridge University Press 2003.

[8] F.J. MacWilliams and N.J.A. Sloane. "Theory of Error-Correcting Codes", North-Holland Publishing Company 1977.

[9] `http://www.inference.phy.cam.ac.uk/mackay/codes/alist.html`

[10] `https://github.com/manuts/gr-ldpc`