Low Density Parity Check Codes

Suayb S. Arslan

February 12, 2015

1 INTRODUCTION

After the birth of information theory in 1948, the concept of information as well as the minimum conditions and requirements for a reliable transmission over noisy channels have been understood very well. Asymptotically (with large block lengths) randomly generated channel codes are shown to achieve the capacity assuming an optimal decoding strategy. However, in the original proof of capacity, the ways and means of efficient design of codes to achieve capacity is absent. In other words, the proof given for the channel capacity was non-constructive. That however later become the main motivation for the research community to find practical codes with efficient encoding and decoding algorithms that shall achieve or come close to the channel capacity.

Low Density Parity Check (LDPC) codes are a subclass of linear codes that can achieve capacity with very large blocklengths. Our previous construction of linear codes were formulated based on algebraic principles and their parity check matrices were often high density. Therefore they are a subclass of High Density Parity Check (HDPC) codes. As we already mentioned in previous notes that the main complexity for such class of codes are due to decoding architectures that are ideally to achieve Maximum Likelihood Decoding (MLD) performance i.e., optimal performance. However the MLD requirement leads to very complicated decoding procedures that makes the practical implementation infeasible. Main attraction of LDPC codes is that the sparse parity check matrices allow us to use suboptimal low complexity decoding algorithms (at most polynomial time) that can achieve near optimal performance. Furthermore, they are found to be suited for implementations that make heavy use of parallelism such as the use of Graphical Processing Unit (GPU) on multiple cores.

2 Graph Terminology

We consider an undirected graph $G(V,E)$ that has vertices contained in the set V and edges in the set E. Two vertices $u,v \in V$ are called adjacent or neighbors if there is an edge $e = \{u,v\}$ connecting u and v. The degree of a vertex v, denoted as $deg(v)$, is the number of edges that are connected to v. For any $G(V,E)$, it is pretty straightforward to show that sum of all degrees of vertices must be an even number. A cycle with c vertices $v_1, v_2, \dots, v_c$, denoted by C_c, is a graph with edges of the form $\{v_1,v_2\},\{v_2,v_3\},\dots,\{v_{c-1},v_c\},\{v_c,v_1\}$. The length of the cycle is defined to be the number of edges one must travel to complete the cycle. An undirected graph $G(V,E)$ may have multiple cycles in it. The minimum length cycle is called the *girth* of the graph G.

A graph $G(V,E)$ is bipartite if V can be partitioned into two nonempty disjoint subsets V_1 and V_2 such that every edge connects a vertex in V_1 and a vertex in V_2. If there is an edge from every vertex in V_1 to every vertex in V_2, $G(V,E)$ is further called complete bipartite graph. The girth of an undirected bipartite graph can only be an even number (do you see why?).

3 LDPC codes

LDPC codes are a subclass of linear block codes that can be obtained from sparse parity check matrices. They are originally invented by Gallager [cite]. For an (n,k) LDPC code, the associated parity check matrix has $r = n-k$ rows and n columns. The rows represent the parity-check equations whereas the columns represent whether the particular location of a codeword is used in the parity check equation or not. If the row weight or the column weights of an LDPC parity check matrix is a fixed number, the code is called regular. An irregular LDPC code relaxes such a constraint on the design of the parity check matrix. A ρ-regular LDPC code is the one where the column weight for each column of the parity check matrix is exactly w_c. This implies that the total number of non-zero entries of the parity check matrix is nw_c. The average row weight is therefore given by $w_r = nw_c/r$. A (w_c,w_r)-regular LDPC code has a parity check matrix whose columns have weight w_c and rows have weight w_r. Usually w_c and w_r are small such that the total number of non-zero entries of the parity check matrix is small. More general definition of sparsity can be given as follows. We finally note that the rate of this code is $1 - w_c/w_r$.

Definition 1: *Let **A** be $r \times n$ parity check matrix defined over $GF(2)$. A sequence of such **A** matrices are called c-sparse if the product (or the size of **A**) "rn" tends to infinity and the number of non-zero elements in $\mathbf{A} < c\max(r,n)$.*

For example, a parity check matrix for a (w_c,w_r)-regular LDPC code is $(w_c+1)-$sparse.

Let us give an example parity check matrix as follows:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}_{4\times 6}$$

Based on our definition of sparsity for parity check matrices, this example constitutes a 3-sparse **H** but not 2-sparse **H**.

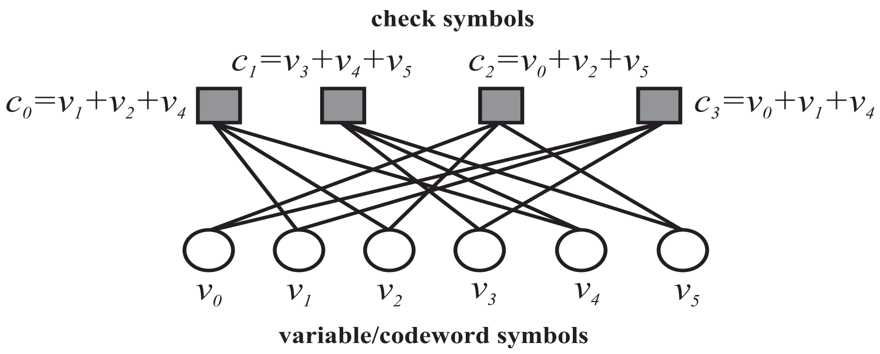

Figure 3.1: A Tanner graph that represents a parity check matrix.

Exercise 1: What is the maximum value that c can take on if we consider only the valid class of LDPC parity check matrices?

A parity check matrix of an LDPC code has a corresponding bipartite graph (a graphical representation – also called Tanner graph) on which the parity check equations are defined. For this example, we can generate the corresponding bipartite graph as shown in Fig. 3.1. Check equations are represented by check nodes or symbols whereas the codeword symbols are represented by variable nodes in the bipartite graph. Note that the entry (i, j) of $\mathbf{H}$ is 1 if and only if the *i*–th check node (c_i) is connected to the *j*–th variable or codeword node (v_j) in the graph. All vectors $\mathbf{c}$ of length six is a valid codeword if $\mathbf{Hc}^T = 0$.

When the parity check matrices are designed for constructing high-performance LDPC codes, the bipartite graphs representing the code play an important role. As will be discussed later, the special properties of these graph representations will guarantee good distance properties and therefore a promising decoding performance. One of the feature of such graphs is their variable and check node degree distributions denoted as $\Lambda(x) = \sum_{d=1}^{d_v} \Lambda_d x^d$ and $\Phi(x) = \sum_{d=1}^{d_c} \Phi_d x^d$ where Λ_d and Φ_d are the probability of having degree-d variable and check nodes, respectively. For example, a (w_c, w_r)-regular LDPC code has $\Lambda(x) = x^{w_r}$ and $\Phi(x) = x^{w_c}$ i.e., check nodes have degree w_r and variable nodes have degree w_c with probability one.

4 ENCODING

The main attraction of LDPC codes is their sparse check matrices which allows very efficient decoding algorithms to perform as well as complex optimal decoders do. We shall see that in the next section. However, encoding complexity of an LDPC code may not be low, as LDPC codes with sparse parity check matrices in general do not need to have sparse generator matrices. Particularly, for large n, the encoding complexity overburden the advantages and benefits offered by low-complexity decoders. In this section, instead of talking about the standard way[1] of obtaining generator matrix from the parity check matrix and large scale

[1]Remember that traditional method is to get $\mathbf{H}$ into a systematic form using row and column permutations and additions to make the right part of $\mathbf{H}$ an identity matrix, from which we can obtain the generator matrix.

matrix multiplications, we consider low-complexity alternatives.

One of the straightforward ways for efficient encoding is called “back substitution". In that method, parities are computed recursively and the over all complexity turns out to be linear in n. One way to allow backward substitution is to design the parity check matrix (deterministic or probabilistic) such that it is in lower triangular form as shown in Fig. 4.1. One example is array LDPC codes [3]. In fact, array codes [4] are designed to correct burst errors and their parity check matrix does not show sparseness unless they are defined over $GF(2)$. Binary array codes can therefore be considered as LDPC codes that can be constructed using deterministic methods. A binary array LDPC code is defined by three parameters: a prime number p and two integers ρ and γ. Its dimensions is $\rho p \times \gamma p$ and given by

$$\mathbf{H}_{array} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \boldsymbol{\alpha} & \boldsymbol{\alpha}^2 & \dots & \boldsymbol{\alpha}^{\gamma-2} & \boldsymbol{\alpha}^{\gamma-1} \\ \mathbf{I} & \boldsymbol{\alpha}^2 & \boldsymbol{\alpha}^4 & \dots & \boldsymbol{\alpha}^{2(\gamma-2)} & \boldsymbol{\alpha}^{2(\gamma-1)} \\ \mathbf{I} & \boldsymbol{\alpha}^3 & \boldsymbol{\alpha}^6 & \dots & \boldsymbol{\alpha}^{3(\gamma-2)} & \boldsymbol{\alpha}^{3(\gamma-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{I} & \boldsymbol{\alpha}^{\rho-1} & \boldsymbol{\alpha}^{2(\rho-1)} & \dots & \boldsymbol{\alpha}^{(\rho-1)(\gamma-2)} & \boldsymbol{\alpha}^{(\rho-1)(\gamma-1)} \end{bmatrix}$$

where $\mathbf{I}$ is the $p \times p$ identity matrix and $\boldsymbol{\alpha}$ is a $p \times p$ column-wise cyclic shift of $\mathbf{I}$ either right or left i.e., for $p = 3$ we have

$$\boldsymbol{\alpha} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \text{ or } \boldsymbol{\alpha} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Exercise 2: Show that ρ and γ provide the column and row weight of the parity check matrix, respectively.

The parity check matrix goes through two additional operations to be put in lower triangular form as shown in Fig. 4.1. The steps can be summarized as follows:

1. The i-th row of $\mathbf{H}_{array}$ is shifted right by $(i-1)$ blocks (p bits) for $i = 1, \dots, j$.

2. The lower triangular elements of the $\rho p \times \rho p$ subblock of shifted $\mathbf{H}_{array}$ is set to nullity to obtain

$$\mathbf{H}_{array} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \boldsymbol{\alpha} & \dots & \boldsymbol{\alpha}^{\rho-2} & \boldsymbol{\alpha}^{\rho-1} & \dots & \boldsymbol{\alpha}^{\gamma-2} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \dots & \boldsymbol{\alpha}^{2(\rho-3)} & \boldsymbol{\alpha}^{2(\rho-2)} & \dots & \boldsymbol{\alpha}^{2(\gamma-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} & \boldsymbol{\alpha}^{(\rho-1)} & \dots & \boldsymbol{\alpha}^{(\rho-1)(\gamma-\rho)} \end{bmatrix}$$

Of course, such modifications to the parity check matrix may degrade the decoding performance heavily. However for high rate constructions with $\rho = 3, 4$, the performance degradation is shown to be minor. Although attractive in its own right, such a parity check matrix construction will degrade the LDPC code performance in general compared to random and unconstrained parity check constructions having the same code parameters.

For a given **H** without being in any special form, bringing it to lower triangular form can be achieved using elementary row operations. However, this is no less complex than the standard/traditional method of encoding. For more efficient encoding, the sparseness of **H** must be preserved as much as possible while triangularization will have to be utilized to allow backward substitution. That is the idea behind [5], as we will describe here shortly. The efficient encoding procedure starts with performing row and column permutations-only step. In that step, the parity check matrix is brought to an approximate lower triangular form as shown in Fig. 4.1. Since all of the operations performed in that step is permutations, the parity check matrix stays sparse.

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}_{r-g\times n-r} & \mathbf{B}_{r-g\times g} & \mathbf{T}_{r-g\times r-g} \\ \mathbf{C}_{g\times n-r} & \mathbf{D}_{g\times g} & \mathbf{E}_{g\times r-g} \end{bmatrix}$$

where $\mathbf{T}_{r-g\times r-g}$ is in lower triangular form where as the rest of the matrices are sparse. The second step of this encoding procedure is the multiplication of H with the following matrix from left,

$$\mathbf{O} = \begin{bmatrix} \mathbf{I}_{r-g} & \mathbf{0} \\ -\mathbf{E}_{g\times r-g}\mathbf{T}^{-1}_{r-g\times r-g} & \mathbf{I}_g \end{bmatrix}$$

Performing this operation, we get

$$\mathbf{OH} = \begin{bmatrix} \mathbf{A}_{r-g\times n-r} & \mathbf{B}_{r-g\times g} & \mathbf{T}_{r-g\times r-g} \\ -\mathbf{E}_{g\times r-g}\mathbf{T}^{-1}_{r-g\times r-g}\mathbf{A}_{r-g\times n-r}+\mathbf{C}_{g\times n-r} & -\mathbf{E}_{g\times r-g}\mathbf{T}^{-1}_{r-g\times r-g}\mathbf{B}_{r-g\times g}+\mathbf{D}_{g\times g} & \mathbf{0} \end{bmatrix}$$

Let $\mathbf{x} = [\mathbf{s} \;\; \mathbf{p}_1 \;\; \mathbf{p}_2]$ be a valid LDPC codeword where **s** is the systematic part, $\mathbf{p}_1$ and $\mathbf{p}_2$ together denote the parity part of the codeword. Therefore, we end up with the following equations,

$$\mathbf{A}\mathbf{s}^T + \mathbf{B}\mathbf{p}_1^T + \mathbf{T}\mathbf{p}_2^T = 0 \tag{4.1}$$

$$(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}+\mathbf{C})\mathbf{s}^T + (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D})\mathbf{p}_1^T = 0 \tag{4.2}$$

However, computational complexity of such a method is qubic in the codeword length n. Since the generator matrix is not in general sparse, the matrix multiplication i.e., encoding is quadratic in n.

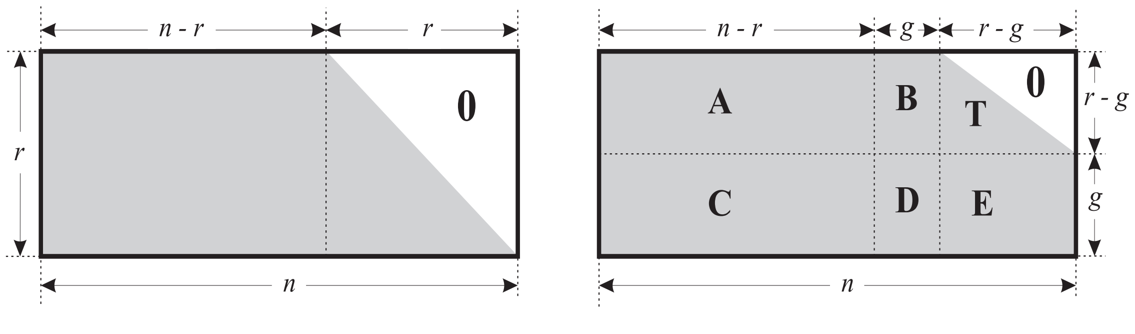

Figure 4.1: Parity check matrices in lower and approximately lower triangular forms [5].

If $(-\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D})$ is singular, by doing some column permutations, it can easily be made non-singular. Thus, the first set of parities can be computed as follows,

$$\mathbf{p}_1^T = (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B}+\mathbf{D})^{-1}(\mathbf{E}\mathbf{T}^{-1}\mathbf{A}-\mathbf{C})\mathbf{s}^T \tag{4.3}$$

As the matrix $\mathbf{T}$ has lower triangular form, we can obtain the second set of parity symbols from backward substitution as follows,

$$\mathbf{p}_2(l) = \sum_{j=1}^{n-r} \mathbf{A}_{l,j}\mathbf{s}_j + \sum_{j=1}^{g} \mathbf{B}_{l,j}\mathbf{p}_1(j) + \sum_{j=1}^{l-1} \mathbf{T}_{l,j}\mathbf{p}_2(j) \tag{4.4}$$

This method is shown to perform almost linear in n (if $g = 0$ the complexity is exactly linear). For large $\mathbf{H}$, bringing an arbitrary sparse matrix in to a form shown in Fig. 4.1 is not straight forward. A greedy algorithm is presented in [5] for an efficient solution to that operation.

5 DECODING

For large parity check matrices, matrix inversions and multiplications are not plausible decoding options. In this section we will explore decoding algorithms that make use of the sparse nature of $\mathbf{H}$. Interestingly, we will see that such low complexity decoding algorithms lead to extremely good performance i.e., near optimal performance.

5.1 MATHEMATICAL FUNDAMENTALS

We begin this section with the following definition.

Definition 1: *Consider the following polynomial of order n with roots* $\boldsymbol{a}_0 = \{a_0, \ldots, a_{n-1}\}$

$$(x-a_0)(x-a_1)(x-a_2)\ldots(x-a_{n-1}) = \sum_{k=0}^{n} s(n,k;\boldsymbol{a}_0)x^k \tag{5.1}$$

where $s(n,k;\boldsymbol{a}_0)$ *are the coefficients of the polynomial obtained after executing the multiplications and arranging the terms in ascending order of powers of x. We define* $s(n,k;\boldsymbol{a}_0)$ *to be the generalized stirling numbers of the first kind.*

Generalized stirling numbers become handy in many problems of combinatorics and part of our discussion of LDPC decoding. More applicable version of such number is known as signless generalized stirling numbers of the first kind, and is given by

$$|s(n,k,\mathbf{a}_0)| = s(n,k,-\mathbf{a}_0) = (-1)^{n-k}s(n,k,\mathbf{a}_0). \tag{5.2}$$

Lemma 1: *Consider a sequence of independent n Bernoulli trials and assume that the probability of success at the ith trial is* $p_i = 1/(1+a_{i-1})$, *with* $a_{i-1} \geq 0, i = 1,2,\ldots,n$. *Let us assume* X_n *is the number of successes up to the nth trial. The probability that* $X_n = k$, *i.e.* $Pr(X_n = k)$ *for* $k = 0,1,\ldots,n$ *is given by*

$$Pr(X_n = k) = \frac{|s(n,k,\boldsymbol{a}_0)|}{(1+a_0)(1+a_1)\ldots(1+a_{n-1})}, k = 0,1,\ldots,n \tag{5.3}$$

where $|s(n,k,\boldsymbol{a}_0)|$ is called the signless stirling number of the first kind and satisfies the triangular recurrence relation given by

$$|s(n,k,\boldsymbol{a}_0)| = |s(n-1,k-1,\boldsymbol{a}_0)| + a_{n-1}|s(n-1,k,\boldsymbol{a}_0)| \tag{5.4}$$

with $s(0,0,\boldsymbol{a}_0)=1$ and $s(n,k,\boldsymbol{a}_0)=0$ for $k>n$.

PROOF: The proof can be found in [2].

Here one can note due to triangular recurrence relation, the probabilities $Pr(X_i = k)$ can efficiently be computed. Next, we explore the sum of the probabilities $Pr(X_n = 2s)$ for $s = 0,1,\ldots,\lfloor n/2 \rfloor$ i.e., the probability that X_n is even. Interestingly, the result is quite clean to be expressed in closed form.

Let us consider the probability generating function

$$G(x) = \sum_{k=0}^{n} Pr(X_i = k)x^k = \sum_{k=0}^{n} \frac{|s(n,k,\mathbf{a}_0)|x^k}{(1+a_0)(1+a_1)\ldots(1+a_{n-1})} \tag{5.5}$$

By definition of $|s(n,k,\mathbf{a}_0)|$, we have

$$G(x) = \prod_{i=1}^{n} \frac{x+a_{i-1}}{1+a_{i-1}} = \prod_{i=1}^{n}(1-p_i+p_i x) \tag{5.6}$$

This probability generating function will be useful for the next lemma.

Lemma 2: *Consider a sequence of independent n Bernoulli trials and assume that the probability of success at the ith trial is p_i for $i = 1,2,\ldots,n$. Let us assume X_n is the number of successes up to the nth trial. The probability that X_n is an even number is given by*

$$\frac{1}{2} + \frac{1}{2}\prod_{i=1}^{n}(1-2p_i) \tag{5.7}$$

PROOF: The following proof is due to Gallager. Consider the probability generator polynomial $G(x)$ and its variant given by Lemma 1,

$$\overline{G}(x) = \prod_{i=1}^{n}(1-p_i-p_i x) \tag{5.8}$$

Once can notice that $\overline{G}(x)$ is identical to $G(x)$ except that odd powers of x are negative. Therefore, adding these two polynomials will yield only the even powers of x but doubled coefficients. If we divide the sum by 2 and letting $x = 1$ will give what is desired. In other words,

$$\frac{G(1)+\overline{G}(1)}{2} = \frac{\prod_{i=1}^{n}(1-p_i+p_i) + \prod_{i=1}^{n}(1-p_i-p_i)}{2} = \frac{1+\prod_{i=1}^{n}(1-2p_i)}{2} \tag{5.9}$$

We can give an alternative proof for this lemma as follows. Let $u(j)$ be the probability that X_j is even for $j \le n$. Note also that $1-u(j)$ is the probability that $X(j)$ is odd. We would like to find $u(n)$. We can express $u(j)$in terms of $u(j-1)$ as follows,

$$\begin{aligned}
u(j) &= u(j-1)(1-p_j)+(1-u(j-1))p_j && (5.10)\\
&= u(j-1)(1-2p_j)+p_j && (5.11)\\
&= u(j-1)(1-2p_j)+p_j-1/2+1/2 && (5.12)\\
u(j)-1/2 &= u(j-1)(1-2p_j)-\frac{1}{2}(1-2p_j) && (5.13)\\
&= \big(u(j-1)-1/2\big)(1-2p_j) && (5.14)
\end{aligned}$$

with the initial condition $u(0)=1$. This is due to the fact that if we have zero bernoulli trials its guaranteed to have even (zero) number of successes. If we let $u'(j)=u(j)-1/2$, we have a simple recurrence

$$u'(j)=u'(j-1)(1-2p_j) \tag{5.15}$$

with $u'(0)=1/2$. Using the initial condition it is not hard to see the general expression

$$u'(n)=u'(0)\prod_{i=1}^{n}(1-2p_i)=\frac{1}{2}\prod_{i=1}^{n}(1-2p_i) \tag{5.16}$$

and therefore we obtain

$$u(n)=\frac{1}{2}+\frac{1}{2}\prod_{i=1}^{n}(1-2p_i) \tag{5.17}$$

as desired. □

5.2 Decoding LDPC codes

The algorithm, which is used to decode LDPC codes, was discovered independently several times in the past and therefore comes under different names. The most common ones are "*the belief propagation algorithm*" , "*the message passing algorithm*" and "*the sum-product algorithm*". We start with some definitions.

5.2.1 Notation and definitions

For simplicity we constrain our consideration to (n,k) LDPC block codes over $GF(2)$. We list the following definitions for convenience that will come handy in our later discussions for LDPC decoding.

- We assume $\mathbf{m}=(m_0,m_1,\ldots,m_{n-1})$ is the message symbols and $\mathbf{r}=(r_0,r_1,\ldots,r_{n-1})$ is the received word (senseword).
- We denote the set of variable nodes that are connected to j-th check node as V_j and set of check nodes that are connected to i-th variable node as C_i.
- We define $\lambda_i=Pr\{v_i=1|\mathbf{r}\}$ to be the conditional probability that $v_i=1$ given the value of $\mathbf{r}$.

- Let $q_{ij}^{(l)}$ be the message sent from variable node v_i to check node c_j at iteration "l". In this context, $q_{ij}^{(l)}(0)$ denotes the amount of belief r_i is zero and $q_{ij}^{(l)}(1)$ denotes the amount of belief r_i is one.
- Let $g_{ji}^{(l)}$ be the message sent from check node c_j to variable node v_i at iteration "l". In this context, $g_{ji}^{(l)}(0)$ denotes the amount of belief r_i is zero and $g_{ji}^{(l)}(1)$ denotes the amount of belief r_i is one.

In order for the belief propagation to work, following rule is respected for relative message passings to be independent.

Definition 2: *(Extrinsic Information Principle) Message sent from a node i along an edge e cannot depend on any message perviously received on edge e.*

Definition 3: *Hyperbolic tangent function (*tanh*) and its inverse (*$\tanh^{-1}$*) can be expressed in the following forms,*

$$\begin{aligned} \tanh(z) &= \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1} && (5.18) \\ \tanh^{-1}(z) &= \frac{1}{2}\ln\left(\frac{1+z}{1-z}\right) \ \textit{for}\ z \in \mathbb{R}, z < 1. && (5.19) \end{aligned}$$

5.2.2 Soft Decoding

We initialize $q_{ij}^{(0)}(1) = \lambda_i$ and $q_{ij}^{(0)}(0) = 1 - \lambda_i$ such that $\sum_{k=1}^{2} q_{ij}^{(0)}(k) = 1$. The rest of the decoding operation consists of two-step procedure; message passing from variable nodes to check nodes and from check nodes to variable nodes.

1. The check nodes compute their messages according to following

$$g_{ji}^{(l)}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_j \setminus i} (1 - 2q_{i'j}^{(l-1)}(1)) \ \text{and}\ g_{ji}^{(l)}(1) = 1 - g_{ji}^{(l)}(0). \qquad (5.20)$$

   where we used the result of Lemma 2 that the probability of having even number of 1's among the variable nodes that are connected to check node c_j except v_i.

2. Next the variable nodes update the message they provide for check nodes using the following equations,

$$\begin{aligned} q_{ij}^{(l)}(0) &= A_{ij}^{(l)}(1-\lambda_i) \prod_{j' \in C_i \setminus j} g_{j'i}^{(l-1)}(0) && (5.21) \\ q_{ij}^{(l)}(1) &= A_{ij}^{(l)}\lambda_i \prod_{j' \in C_i \setminus j} g_{j'i}^{(l-1)}(1) && (5.22) \end{aligned}$$

   where $A_{ij}^{(l)}$ is chosen such that $\sum_{k=1}^{2} q_{ij}^{(l)}(k) = 1$.

During the execution of the second step, variable nodes update their estimated value of r_i. this step no different than the second step except it includes the information coming from j-th check node c_j as well. This is given by

$$\hat{\lambda}_i^{(l)}(0) = A_i^{(l)}(1-\lambda_i)\prod_{j'\in C_i} g_{j'i}^{(l)}(0) \tag{5.23}$$

$$\hat{\lambda}_i^{(l)}(1) = A_i^{(l)}\lambda_i\prod_{j'\in C_i} g_{j'i}^{(l)}(1) \tag{5.24}$$

from which the decision rule can simple be stated as,

$$\hat{m}_i^{(l)} = \begin{cases} 1, & \text{if } \hat{\lambda}_i^{(l)}(0) < \hat{\lambda}_i^{(l)}(1). \\ 0, & \text{otherwise.} \end{cases} \tag{5.25}$$

The decoder iteratively goes back to step 1 and increments $l \to l+1$ and recomputes the variables until one of the two conditions hold: Either the vector $(m_0^{(l)}, m_1^{(l)}, \ldots, m_{n-1}^{(l)})$ satisfies all the parity check equations or a predetermined maximum number of iterations is reached. As can be seen, for large block lengths the product computation might be costly to implement. In addition, multiplication of tiny numbers or very large numbers will cause numerical instabilities which one needs to avoid in any software or hardware implementation. Likelihoods or logarithmic likelihood avoids instability problems by mapping $[-1,1]$ interval to $[-\infty,+\infty]$ and transforms all multiplications to additions.

Let us define the logarithmic likelihood ratio,

$$\Omega_i^{(0)} = \ln\left(\frac{Pr\{v_i = 0|\mathbf{r}\}}{Pr\{v_i = 1|\mathbf{r}\}}\right) = \ln\left(\frac{1-\lambda_i}{\lambda_i}\right) \tag{5.26}$$

From this relationship, we can deduce $\lambda_i = 1/(1+e^{\Lambda_{ij}^{(0)}})$. We use the same initilization values $q_{ij}^{(0)}(1) = \lambda_i$ and $q_{ij}^{(0)}(0) = 1-\lambda_i$ such that $\sum_{k=1}^{2} q_{ij}^{(0)}(k) = 1$. Now we update the steps of decoding algorithm as follows,

1. The check nodes compute their messages according to following

$$\Lambda_{ji}^{(l)} = \ln\left(\frac{g_{ji}^{(l)}(0)}{g_{ji}^{(l)}(1)}\right) = \ln\left(\frac{1+\prod_{i'\in V_j\backslash i}(1-2q_{i'j}^{(l-1)}(1))}{1-\prod_{i'\in V_j\backslash i}(1-2q_{i'j}^{(l-1)}(1))}\right) \tag{5.27}$$

$$= \ln\left(\frac{1+\prod_{i'\in V_j\backslash i}\tanh(\frac{\Lambda_{i'j}^{(l-1)}}{2})}{1-\prod_{i'\in V_j\backslash i}\tanh(\frac{\Lambda_{i'j}^{(l-1)}}{2})}\right) \tag{5.28}$$

$$= 2\tanh^{-1}\left(\prod_{i'\in V_j\backslash i}\tanh\left(\frac{\Lambda_{i'j}^{(l-1)}}{2}\right)\right) \tag{5.29}$$

where we used Definition 2 and the fact that,

$$\Lambda_{ij}^{(l)} = \ln\left(\frac{q_{ij}^{(l)}(0)}{q_{ij}^{(l)}(1)}\right) = \ln\left(\frac{1-q_{ij}^{(l)}(1)}{q_{ij}^{(l)}(1)}\right) \Rightarrow q_{ij}^{(l)}(1) = 1/(1+e^{\Lambda_{ij}^{(l)}}) \tag{5.30}$$

$$\Rightarrow 1-2q_{ij}^{(l)}(1) = \frac{e^{\Lambda_{ij}^{(l)}}-1}{e^{\Lambda_{ij}^{(l)}}+1} = \tanh\left(\frac{\Lambda_{ij}^{(l)}}{2}\right) \tag{5.31}$$

2. Next the variable nodes update the message they provide for check nodes using the rule,

$$\begin{aligned}\Lambda_{ij}^{(l)} = \ln\left(\frac{q_{ij}^{(l)}(0)}{q_{ij}^{(l)}(1)}\right) &= \ln\left(\frac{1-\lambda_i}{\lambda_i}\prod_{j'\in C_i\backslash j}\frac{g_{j'i}^{(l-1)}(0)}{g_{j'i}^{(l-1)}(1)}\right) &(5.32)\\ &= \Omega_i^{(0)} + \sum_{j'\in C_i\backslash j}\Lambda_{j'i}^{(l-1)} &(5.33)\end{aligned}$$

The decision rule will be based on the metric $\Omega_i^{(l)}$ computed at the l-th iteration as follows,

$$\Omega_i^{(l)} = \ln\left(\frac{\hat{\lambda}_i^{(l)}(0)}{\hat{\lambda}_i^{(l)}(1)}\right) = \Omega_i^{(0)} + \sum_{j'\in C_i}\Lambda_{j'i}^{(l)} \tag{5.34}$$

according to following rule,

$$\hat{m}_i^{(l)} = \begin{cases} 1, & \text{if } \Omega_i^{(l)} < 0. \\ 0, & \text{otherwise.}\end{cases} \tag{5.35}$$

We note that tanh(.) and $\tanh^{-1}$(.) operations might be costly from an implementation point of view. Therefore further simplifications are devised for the real implemented BP algorithm in many applications. For example in one of them, check node's computations are done according to the following rule (instead of step 1),

$$\Lambda_{ji}^{(l)} \approx \min_{i'\in V_j\backslash i}(\{\Lambda_{i'j}^{(l-1)}\})\left(\prod_{i'\in V_j\backslash i} sgn\left(\Lambda_{i'j}^{(l-1)}\right)\right) \tag{5.36}$$

which is based on the approximation $\ln(\cosh(x)) \approx |x| - \ln(2)$ for $x >> 1$. Some performance loss may be incurred due to this approximation yet it is easy to justify that using such an approximation the check node rule computation complexity can dramatically be reduced.

5.2.3 Erasure and Hard error Decoding

Erasure Decoding: BP algorithm greatly simplifies when the channel is a binary erasure channel i.e., the channel does not introduce errors but erases packets or bits [7]. We assume $r_i \in \{-1, 0, +1\}$ where 0 denotes an erasure. The decoding algorithm executes the following steps in order,

- We initialize $\hat{m}_i^{(0)} = r_i$ and $l = 0$.
- We set $\Lambda_{ij}^{(l)} = \hat{m}_i^{(l)}$ and $l = l + 1$.
- The check nodes compute their messages according to following $\Lambda_{ji}^{(l)} = \prod_{i' \in V_j \setminus i} \Lambda_{i'j}^{(l-1)}$. This means that if all incoming messages are nonzero, either $+1$ or -1 is sent to a variable node that checks the parity equation, otherwise a zero is sent.
- If for variable node s, if any $\Lambda_{js}^{(l)}$ coming from the check nodes in C_s is nonzero, we set $\hat{m}_i^{(l)} = \Lambda_{js}^{(l)}$ and i-th variable node is said to be known. Otherwise, we do not decode.
- If $\hat{m}_i^{(l)} \neq 0$ for all i, stop. Otherwise, go back to step 2.

Since it is erasure channel we are dealing with, once the value of the message bit is determined in any iteration of the BP algorithm, its value need not to be updated anymore mainly because only reliable (no errors are assumed in an erasure correcting mode) information is communicated between nodes.

Finally we note an interesting use of BP algorithm for erasure decoding. If we assume the parity check matrix can be split into two matrices $\mathbf{H} = [\mathbf{H}_m | \mathbf{H}_p]$ such that $\mathbf{H}_p$ is in low triangular form, then BP algorithm can be used to do the encoding (same operation of back substitution introduced earlier). This can be done by assuming the values of the first k variable nodes to have the k values of the message symbols. The parities are assumed to be erasures. Using the check equations, the BP algorithm finds the parity values with probability one due to the low triangular structure at most k iterations. Advantages of such a scheme could be

- Linear time complexity in k.
- The encoder and decoder share the same hardware and hence save space.

HARD ERROR DECODING: One of the most popular channel model is known as the Binary Symmetric Channel (BSC) in which one of the two possible input bits are sent through a channel and one of the two possible bits are received with an error probability ϵ_0. The channel is therefore to introduce only errors but not erasures i.e., $r_i \in \{-1, +1\}$. The decoding algorithm executes the following steps in order[2] for this channel,

- We initialize $\hat{m}_i^{(0)} = r_i$ and $l = 0$.
- We set $\Lambda_{ij}^{(l)} = \hat{m}_i^{(l)}$ and then $l = l + 1$.
- The check nodes compute their messages according to following $\Lambda_{ji}^{(l)} = \prod_{i' \in V_j \setminus i} \Lambda_{i'j}^{(l-1)}$. This means that either $+1$ or -1 is sent to a variable node to check the parity equation.

[2] This algorithm is the most basic decoding scheme primarily introduced in [1], and named as Gallager decoding algorithm A. Later, slightly different improvements have been made to it.

- For variable node s, we consider the set $\{\Lambda_{j_1 s}^{(l)}, \Lambda_{j_2 s}^{(l)}, \ldots, \Lambda_{j_{|C_s|} s}^{(l)}, \hat{m}_i^{(l)}\}$ where $\{j_1, j_2, \ldots, j_{|C_s|}\}$ are the indexes of the check nodes in C_s. We set

$$\begin{aligned} \hat{m}_i^{(l)} &= \text{Majority}\left(\Lambda_{j_1 s}^{(l)}, \Lambda_{j_2 s}^{(l)}, \ldots, \Lambda_{j_{|C_s|} s}^{(l)}, \hat{m}_s^{(l)}\right) && (5.37) \\ &= \left\lfloor 2 + \frac{\sum_{t=1}^{|C_s|} \Lambda_{j_t s}^{(l)} + \hat{m}_i^{(l)}}{|C_s| + 1} \right\rfloor_{even} - 1 && (5.38) \end{aligned}$$

  where $\lfloor . \rfloor_{even}$ rounds down to the nearest even integer. Note that the value of $\hat{m}_i^{(l)}$ is flipped if $\lceil |C_s|/2 \rceil + 1$ or more elements of $\{\Lambda_{j_1 s}^{(l)}, \Lambda_{j_2 s}^{(l)}, \ldots, \Lambda_{j_{|C_s|} s}^{(l)}\}$ disagree with $\hat{m}_s^{(l)}$.

- If $\hat{m}_i^{(l)} = \hat{m}_i^{(l-1)}$ for all i, stop. Otherwise, go back to step 2.

Exercise 3: Show that the equation (5.38) is equivalent to Majority logic given in equation (5.37).

6 Analysis of LDPC codes: Density Evolution (DE)

6.1 DE for Binary Erasure Channel

Let us start with LDPC codes over binary erasure channels (BEC) as it is easier to illustrate the concept of “Density Evolution" over such channels. The idea behind density evolution is the predict the ultimate error/erasure correction capability of an LDPC code. Since the extrinsic messages (various realizations of many random variables) are passed from check nodes to message nodes and back to check nodes, it may be possible that message originated from a node is received after a few rounds of the algorithm because the code may contain cycles in it. However in this case, the “independent message" transfer will be violated. In the method of Density Evolution, messages are assumed to be independent (which is approximately true for large girth, long block length LDPC codes. If the messages are independent i.e., the Tanner graph representation of the LDPC code contains no cycles of some length $2f$, then any node can be the root of a tree with l branches consisting of variable and check nodes. Under such assumptions, the propagation of erasure probabilities (density) are of our major concern as the algorithm evolve though iterations.

By a slight abuse of the notation for $r_i \in \{0,1\}$ where a zero denotes an erasure (or an unknown) as before and one denotes either $\{+1\}$ or $\{-1\}$, indicating that the value of a variable node is known. We observe that the output of a check node j (intended for variable node i) is only known if all variable nodes $i' \in V_j \backslash i$ are nonzero. This means that a check node performs the *AND* logic operation. Similarly, the output of a variable node i is known if at least one check node $j' \in C_i \backslash j$ is nonzero. This means that a variable node performs the *OR* logic operation.

We generate a tree by first choosing an edge from the graph. Let the probability of a randomly chosen edge is connected to degree-d variable node and a degree-d check node be λ_d and ρ_d, respectively. Therefore, we can talk about edge perspective degree distributions

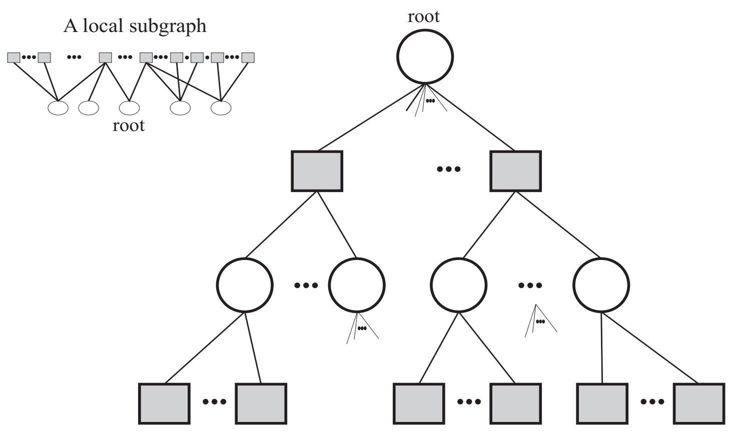

Figure 6.1: A subgraph of the bipartite representation of an LDPC code asymptotically approaches to a tree, which validates the assumption under which the density evolution formulas have been derived.

defined as

$$\lambda(x) = \sum_{d=1}^{d_v} \lambda_d x^{d-1} \quad \text{and} \quad \rho(x) = \sum_{d=1}^{d_c} \rho_d x^{d-1} \tag{6.1}$$

Exercise 4: Show that the relationship between edge-perspective and node-perspective degree distributions is given by

$$\lambda(x) = \frac{\Lambda'(x)}{\Lambda'(1)} \qquad \rho(x) = \frac{\Phi'(x)}{\Phi'(1)} \tag{6.2}$$

where $\Lambda'(x)$ and $\Phi'(x)$ are formal derivatives of $\Lambda(x)$ and $\Phi(x)$.

Exercise 5: Show that the rate of the LDPC code R can be computed by

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} \tag{6.3}$$

Let $P_v^{(l)}$ and $P_c^{(l)}$ be the probability of density of erasures at the variable and check nodes at the l-th iteration of the BP algorithm. For any LDPC code with edge perspective degree distributions $\lambda(x)$ and $\rho(x)$ used over a BEC characterized with the erasure probability ϵ_0, the following recursive relations can be established using the $AND-OR$ argument,

$$P_c^{(l)} = 1 - \sum_{i=1}^{d_c} \rho_i \left(1 - P_v^{(l)}\right)^{i-1} = 1 - \rho(1 - P_v^{(l)}) \tag{6.4}$$

$$P_v^{(l+1)} = \epsilon_0 \sum_{i=1}^{d_v} \lambda_i \left(P_c^{(l)}\right)^{i-1} = \epsilon_0 \lambda(P_c^{(l)}) = \epsilon_0 \lambda(1 - \rho(1 - P_v^{(l)})) \tag{6.5}$$

From these recursions, it is apparent that for $l \to \infty$ error free decoding is possible if and only if $P_v^{(l+1)} < P_v^{(l)}$. This yields the condition that

$$\epsilon_0 \lambda(1 - \rho(1 - x)) < x \quad \text{for} \quad x \in (0, \epsilon_0] \tag{6.6}$$

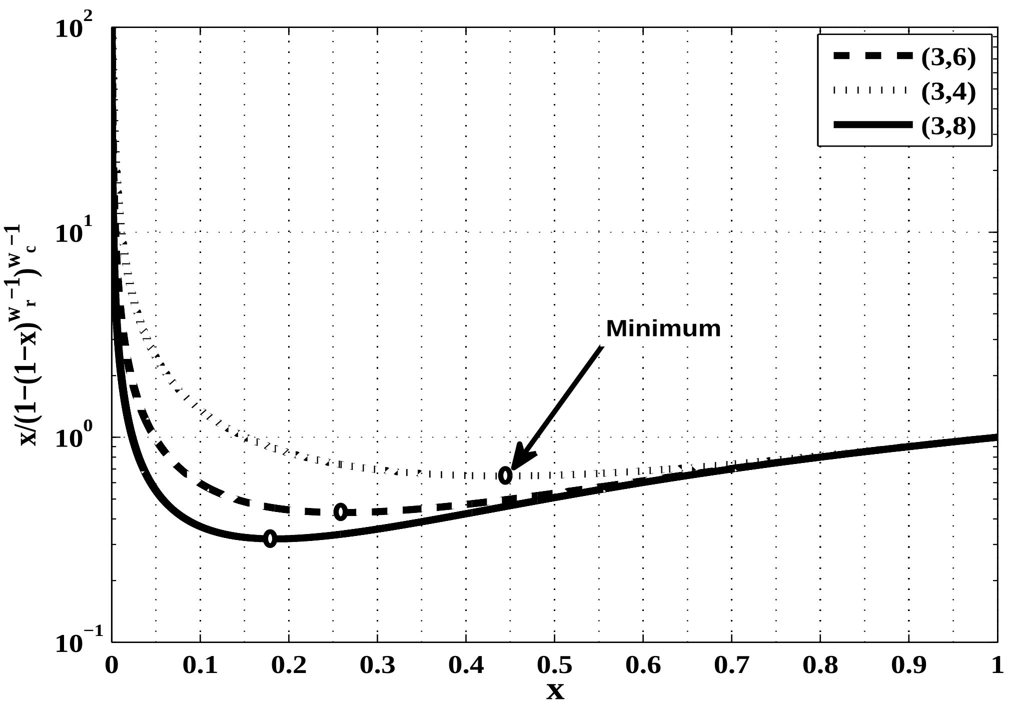

Figure 6.2: Supremum of can be found by minimizing $\frac{x}{\lambda(1-\rho(1-x))}$. Shown are there examples for three different regular LDPC codes.

So the interesting question is to ask the supremum of all the set of ϵ_0 values that satisfy the above inequality. This is the maximal erasure probability of the channel below which error-free decoding shall be possible. Let us denote this value ϵ_0^* and it is given by

$$\epsilon_0^* = \sup\{\epsilon_0 : \epsilon_0 \lambda(1-\rho(1-x)) < x, \forall x, x \in (0, \epsilon_0]\} \tag{6.7}$$

In coding community, ϵ_0^* is known as the decoding threshold. It is the maximal fraction of erasures an LDPC code with given edge degree distributions can correct using the BP algorithm. However, there is an equivalent way of expressing ϵ_0^* as follows,

$$\epsilon_0^* = \inf\left\{\frac{x}{\lambda(1-\rho(1-x))}, \forall x, x \in (0,1)\right\} \tag{6.8}$$

For a regular code, this optimization can be solved for a closed form expression. The details can be found in [6]. Let us consider three sample $(3,8)$, $(3,6)$ and $(3,4)$ regular LDPC codes. Note that these codes have code rates $(1-w_c/w_r)$ 5/8, 1/2 and 1/4, respectively. The x values that minimize $\frac{x}{\lambda(1-\rho(1-x))}$ are ≈ 0.1844, ≈ 0.2606 and ≈ 0.4417, respectively. Thus, if we evaluate it at these optimal values, we can get $\epsilon_0^* = 0.3193$, $\epsilon_0^* = 0.4294$ and $\epsilon_0^* = 0.6359$. These minimums are plotted in Fig. 6.2 for reference. From our previous notes, we recall that the capacity of a BEC is $1-R$ where R is the rate of the optimal code (erasure rate of the channel). Therefore, we note that $(3,8)$ regular LDPC is off the capacity by $0.375-0.3193=0.0557$, $(3,6)$ regular LDPC is off the capacity by $0.5-0.4294=0.0706$ and $(3,4)$ regular LDPC is off the capacity by $0.75-0.6359=0.1141$. As we can see, higher LDPC codes have better potential for achieving the capacity as predicted by our previous general treatment of linear codes. Furthermore, it is shown that irregular LDPC codes have better decoding thresholds than do the regular LDPC codes.

If the erasure probability ϵ_0 of the BEC is greater than the threshold ϵ_0^* i.e., $\epsilon_0 > \epsilon_0^*$, then the BP decoding will not succeed and there will always remain a fixed fraction of erasures that cannot be corrected [8]. Since this fraction is a function of ϵ_0, we start by defining the following for $\epsilon_0 > \epsilon_0^*$

$$x(\epsilon_0) = \sup\{x : \epsilon_0 \lambda(1-\rho(1-x)) = x, \forall x, x \in (0,\epsilon_0]\} \tag{6.9}$$

which is the emitted erasure probability from variable nodes when the iterations terminate. Corresponding messages by the check nodes is $1-\rho(1-x(\epsilon_0))$. For a variable node to be erasure, all of the incoming information as well as the current value is an erasure. This happens with probability $\epsilon_0\Lambda(1-\rho(1-x(\epsilon_0)))$. Therefore, the asymptotical performance is given in a parametric form given by

$$\Big(\epsilon_0, \epsilon_0\Lambda(1-\rho(1-x(\epsilon_0)))\Big) \quad \text{for} \quad \epsilon_0 \geq \epsilon_0^* \tag{6.10}$$

6.2 DE for Binary Symmetric Channel

Using similar notation to the previous subsection, let $P_v^{(l)}$ be the error probability at the variable node at iteration l. Note that we have the initialization $P_v^{(0)} = \epsilon_0$, i.e., the cross over probability of the BSC. According to the hard decision decoding algorithm introduced in the previous section, a check symbol computes the wrong check symbol $\in \{+1,-1\}$ if an only if an odd number of errors are passed from its neighbors. Conditioning on the check node having a node degree d, this probability (using the result of Lemma 2) is given by

$$P_{c|d}^{(l)} = \frac{1-(1-2P_{v|d}^{(l)})^{d-1}}{2} \tag{6.11}$$

where $|d$ denotes the conditioning operation. If we average over the degree distribution, it is not hard to see that this probability becomes,

$$P_c^{(l)} = \frac{1-\rho(1-2P_v^{(l)})}{2} \tag{6.12}$$

Consider the variable node i conditioned on having degree d (assuming all incoming check nodes to carry independent information, which is in fact the main basis of DE). Assume that this node has wrong value at the lth iteration, the probability that it flips its value (i.e., changes it to the correct value) is given by (i.e., $\lceil d/2 \rceil$ or less check node information agree with the variable node value)

$$P_v^{(l)} \sum_{t=0}^{\lceil d/2 \rceil} \binom{d}{t} \left(\frac{1-\rho(1-2P_v^{(l)})}{2}\right)^t \left(\frac{1+\rho(1-2P_v^{(l)})}{2}\right)^{d-t} = P_v^{(l)} \psi(t, d-t) \tag{6.13}$$

Similarly assume that this node has the correct value at the lth iteration, the probability it flips its value (i.e., changes it to the wrong value) is given by (i.e., $\lceil d/2 \rceil$ or less check node information agree with the variable node value)

$$(1-P_v^{(l)})\sum_{t=0}^{\lceil d/2\rceil}\binom{d}{t}\left(\frac{1+\rho(1-2P_v^{(l)})}{2}\right)^t\left(\frac{1-\rho(1-2P_v^{(l)})}{2}\right)^{d-t} = (1-P_v^{(l)})\psi(d-t,t) \qquad (6.14)$$

Since the events are assumed to be independent of eachother, we have conditional probability in the next iteration expressed as,

$$P_{v|d}^{(l+1)} = P_v^{(l)} - P_v^{(l)}\psi(t,d-t) + (1-P_v^{(l)})\psi(d-t,t) \qquad (6.15)$$

The unconditional expression will be given by averaging over the distribution $\lambda(x)$, i.e.,

$$P_v^{(l+1)} = P_v^{(l)} - \sum_{d=1}^{d_v}\lambda_d\left(P_v^{(l)}\psi(t,d-t) - (1-P_v^{(l)})\psi(d-t,t)\right) \qquad (6.16)$$

7 Graph representations and Decoding performance

The properties of the Tanner graph that describes an LDPC code determines the performance of the code. A cycle in a Tanner graph refers to a finite set of connected edges, the edge starts and ends at the same node, and it satisfies the condition that no node (except the initial and final node) appears more than once. The length of a cycle is simply the number of edges of the cycle. The girth of a Tanner graph is the length of the shortest cycle in the graph. Girths in the Tanner graphs of LDPC codes may prevent the belief propagation decoding algorithm from converging. Further, cycles, especially short cycles, degrade the performance of LDPC decoders, because they affect the independence of the extrinsic information exchanged in the iterative decoding.

To be continued...

REFERENCES

[1] R. G. Gallager, “Low-density parity-check codes," IRE Transactions on Information Theory, vol. 8, no. 1, pp. 21–28, 1962

[2] C. A. Charalambides, “Combinatorial methods in Discrete distributions," John Willet & Sons, Inc. 2005.

[3] E. Eleftheriou and S. Olcer, “Low-density parity–check codes for digital subscriber lines," inProc. IEEE Int. Conf. Communications (ICC 2002), vol. 3, 2002, pp. 1752–1757.

[4] M. Blaum, P. Farrell, and H. van Tilborg, “Array codes" in Handbook of Coding Theory, V. S. Pless and W. C. Huffman Eds., Elsevier 1998.

[5] T. Richardson and R. Urbanke, “Efficient encoding of low-density parity-check codes," IEEE Trans. Inform. Theory, vol. 47, pp. 638–656, Feb. 2000.

[6] C. Schlegel and L. Perez, Trellis and Turbo Coding, IEEE/Wiley, Piscataway, NJ, USA, 2004.

[7] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, “Practical loss-resilient codes," *in Proc. 29th Annu. ACM Symp. Theory of Computing*, 1997, pp. 150–159.

[8] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, “Finite length analysis of low-density parity-check codes," IEEE Trans. on Information Theory, pp. 1570– 1579, June 2002.