Low Density Parity Check (LDPC) Codes

Low Density Parity Check Codes were invented in 1963 by R. G. Gallager. In addition to suggesting the use of codes with sparse parity check matrices, Gallager suggested an iterative decoding algorithm (message-passing decoders) and showed that using this type of decoder, one can come close to Shannon's bounds.

In general, an LDPC code is the null space of a sparse (low-density) matrix H, i.e.,

$$\underline{v} H^T = \underline{0}$$

where H^T is a low-density matrix in the following sense:

Assume that H has J rows and n columns, and there are (on the average) i ones in the columns and (on the average) l 1's on rows. In $i \ll J$ and $l \ll n$, we call the matrix H low-density or sparse.

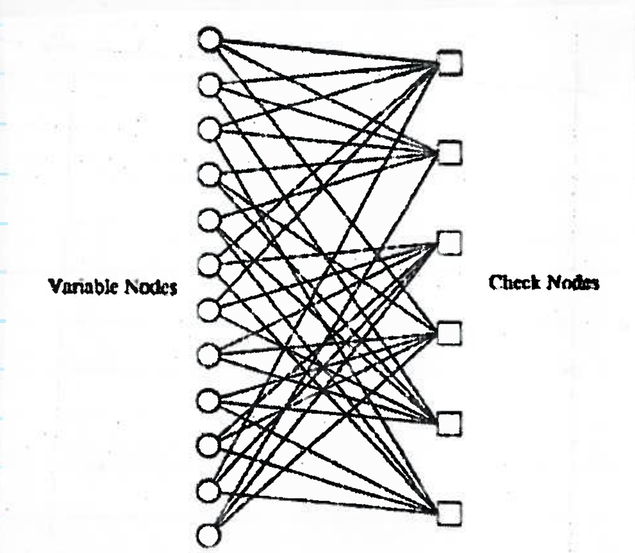
In the regular or Gallager LDPC Codes, the number of 1's ~~on~~ in each column or on each row are the same.

Example: (3, 6) Regular LDPC Code

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

This matrix has 3 one's in each column and 6, 1's on each row.

Graphically, LDPC Code can be represented bi-partite graphs as suggested by Tanner.



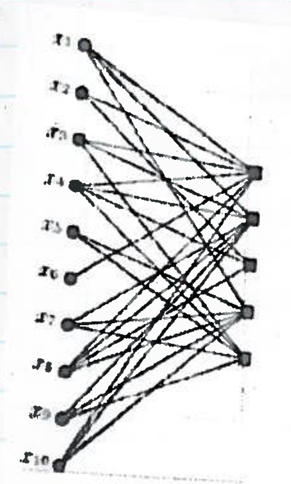
Graphical representation of a (3, 6)-regular LDPC code of length 12. The left nodes represent the variable nodes whereas the right nodes represent the check nodes.

On one side are the message nodes also called variable nodes; on the other side of the graph are constraint nodes or check nodes.

A legitimate pattern, i.e., a codeword is a bit stream that when fed to variable nodes, the check nodes will all have value zero.

The abov code is a regular LDPC code since each node on the left is incident by 3 edges and each node on right receives 6 edges. We say that message nodes have degree 3 and check nodes have degree 6.

The code whose Tanner graph is shown here is an irregular LDPC Code. Nodes $x_1, x_3, x_7; x_9, x_{10}$ have degree 3. Nodes x_2, x_5 have degree 2. Nodes x_4 and x_8 have degree 4 and x_6 has degree one.



Check nodes have degrees 7, 7, 3, 6, 5.

An LDPC code is specified in terms of an edge degree distribution for variable nodes and anoth degree distribution for check nodes.

Let λ_i be the fraction of edges that enter/exit variable nodes of degree i. Define degree distribution polynomial:

$$\lambda(x) = \sum_{i \geq 1} \lambda_i x^{i-1}$$

It is clear that $\lambda(1) = \sum_{i \geq 1} \lambda_i = 1$

For the above example:

$$\lambda(x) = \frac{1}{28} + \frac{1}{7} x + \frac{15}{28} x^2 + \frac{2}{7} x^3$$

$\lambda_1 = \frac{1}{28}$ since only 1 of 28 edges is incident on an edge of degree 1.

$\lambda_2 = \frac{4}{28} = \frac{1}{7}$ since 2×2 edges fall upon two nodes of degree 2. Similarly, λ_3 and λ_4 are found to be $\frac{15}{28}$ and $\frac{2}{7}$, respectively.

$$\int_0^1 \lambda(x)\, dx = \sum_{i \geq 1} \frac{\lambda_i}{i} x^i \Big|_0^1 = \sum_{i \geq 1} \frac{\lambda_i}{i}.$$

In a similar, way, a degree distribution $\rho(x)$ can be defined for the check nodes:

$$\rho(x) = \sum_{i \geq 1} \rho_i x^{i-1}$$

where ρ_i is the fraction of edges incident on a check node of degree i.

The rate of a (λ, ρ) Code is given by

$$\rho(\lambda, \rho) = 1 - \frac{\int \rho}{\int \lambda}$$

where integrals are taken from 0 to 1.

Rate of the regular LDPC Code:

Take the example of the $(3,6)$ Code discussed above. Since all variable nodes are of degree 3 then $\lambda_3 = 1$ and $\lambda(x) = x^2$.

Similarly $\rho_6 = 1 \Rightarrow \rho(x) = x^5$

$$\int_0^1 \lambda(x)\,dx = \frac{1}{3} \quad \text{and} \quad \int_0^1 \rho(x)\,dx = \frac{1}{6}$$

So, the rate is

$$r = 1 - \frac{\int \rho}{\int \lambda} = 1 - \frac{1/6}{1/3} = \frac{1}{2}$$

Assignment: Find the rate of the irregular Code discussed above (graph of page 9-3).

Encoding of LDPC Codes

While sparsity of the check matrix makes the decoding of LDPC Codes, the fact that they are defined in terms of parity check matrix makes their encoding complex.

Now it is a good time to reflect on the question of why we prefer cyclic codes and systematic codes. If a linear code is not cyclic, we need to find codewords by multiplying the information vector $\underline{u}$ by $\underline{G}$. It means n vector multiplications (as the number of columns of G is n). It also is evident that for each vector multiplication, we need on the average $n/2$ operations (say XOR and add). So, the complexity is $O(n^2)$. For a cyclic code the complexity is $O(n)$, i.e., it is linear in n. For a non-cyclic but systematic code, we need to find $(n-k)$ parities each requiring (on the average $\frac{k}{2}$ operation. So, the order of encoding is $O(nk)$.

For LDPC Codes encoding is difficult since the graph can only show whether a bit pattern is a codeword or not. It cannot be used for relating the messages to codewords.

To ease encoding there are several different approaches:

- To use cascaded rather than bi-partite graphs. This means doing encoding in several stages. By choosing the number and size of the stages, one can design codes that are encodeable and decodable in linear time.

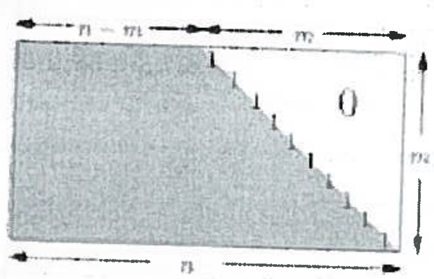
The disadvantage of this technique is that since each stage adds parity to the message and parity from previous stage. The length of data to the total codeword length is small (low rate). This results in performance loss compared to a standard LDPC code.

- The other approach is to use codes that have lower triangular form. This is similar to solving system of linear equations using Gauss elimination.

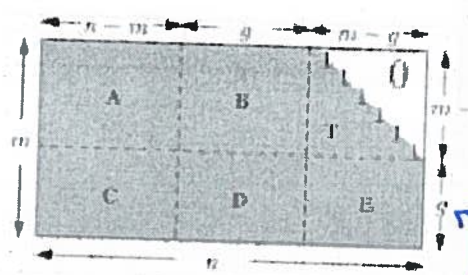
This approach while guarantees linear time encoding complexity, results in some loss of performance due to being restricted to a class of LDPC codes.

- Starting from a standard LDPC code, we try to make its parity check matrix lower triangular and stop when you cannot go further (Richardson and Urbanke).

This results in an approximate lower triangular matrix



An equivalent parity-check matrix in lower triangular form.



The parity-check matrix in approximate lower triangular form.

Then it is shown that encoding complexity is $O(n+g^2)$ where g is the gap.

EFFICIENT ENCODERS BASED ON APPROXIMATE LOWER TRIANGULATIONS

In this section, we shall develop an algorithm for constructing efficient encoders for LDPC codes. The efficiency of the encoder arises from the sparseness of the parity-check matrix H and the algorithm can be applied to any (sparse) H. Although our example is binary, the algorithm applies generally to matrices H whose entries belong to a field F. We assume throughout that the rows of H are linearly independent. If the rows are linearly dependent, then the algorithm which constructs the encoder will detect the dependency and either one can choose a different matrix H or one can eliminate the redundant rows from H in the encoding process.

Assume we are given an $m \times n$ parity-check matrix H over F. By definition, the associated code consists of the set of n-tuples x over F such that

$$Hx^T = 0^T.$$

Probably the most straightforward way of constructing an encoder for such a code is the following. By means of Gaussian elimination bring H into an equivalent lower triangular form as shown in Fig. 2. Split the vector x into a *systematic* part s, $s \in F^{n-m}$, and a *parity* part p, $p \in F^m$, such that $x = (s, p)$. Construct a *systematic* encoder as follows: i) Fill s with the $(n - m)$ desired information symbols. ii) Determine the m parity-check symbols using *back-substitution.* More precisely, for $l \in [m]$ calculate

$$p_l = \sum_{j=1}^{n-m} H_{l,j} s_j + \sum_{j=1}^{l-1} H_{l,j+n-m} p_j.$$

What is the complexity of such an encoding scheme? Bringing the matrix H into the desired form requires $O(n^3)$ operations of *preprocessing*. The actual encoding then requires $O(n^2)$ operations since, in general, after the preprocessing the matrix will no longer be sparse. More precisely, we expect that we need about $n^2 \frac{r(1-r)}{2}$ XOR operations to accomplish this encoding, where r is the rate of the code.

Given that the original parity-check matrix H is sparse, one might wonder if encoding can be accomplished in $O(n)$. As we will show, typically for codes which allow transmission at rates close to capacity, linear time encoding is indeed possible. And for those codes for which our encoding scheme still leads to quadratic encoding complexity the constant factor in front of the n^2 term is typically very small so that the encoding complexity stays manageable up to very large block lengths.

Our proposed encoder is motivated by the above example. Assume that by *performing row and column permutations only* we can bring the parity-check matrix into the form indicated in Fig. 3. We say that H is in *approximate lower triangular form.* Note that since this transformation was accomplished solely by permutations, the matrix is still sparse. More precisely, assume that we bring the matrix in the form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} \tag{5}$$

where A is $(m-g)\times(n-m)$, B is $(m-g)\times g$, T is $(m-g)\times(m-g)$, C is $g\times(n-m)$, D is $g\times g$, and, finally, E is $g\times(m-g)$. Further, all these matrices are sparse[2] and T is lower triangular with ones along the diagonal. Multiplying this matrix from the left by

$$\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix} \tag{6}$$

we get

$$\begin{pmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{pmatrix}. \tag{7}$$

Let $x = (s, p_1, p_2)$ where s denotes the systematic part, p_1 and p_2 combined denote the parity part, p_1 has length g, and p_2 has length $(m - g)$. The defining equation $Hx^T = 0^T$ splits naturally into two equations, namely

$$As^T + Bp_1^T + Tp_2^T = 0 \tag{8}$$

and

$$(-ET^{-1}A + C)s^T + (-ET^{-1}B + D)p_1^T = 0. \tag{9}$$

Define $\phi := -ET^{-1}B + D$ and assume for the moment that ϕ is nonsingular. We will discuss the general case shortly. Then from (9) we conclude that

$$p_1^T = -\phi^{-1}(-ET^{-1}A + C)s^T.$$

Hence, once the $g \times (n - m)$ matrix $-\phi^{-1}(-ET^{-1}A + C)$ has been precomputed, the determination of p_1 can be accomplished in complexity $O(g \times (n - m))$ simply by performing

[2]More precisely, each matrix contains at most $O(n)$ elements.

For example for the (3,6) LDPC Code H can be transformed into:

$$H = \left(\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right)$$

$$= \left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right).$$

1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 8, 9

by column re-ordering. This is an approximate lower triangular matrix with $g = 2$.

Then E can be made into $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ by Gauss elimination

$$\left[\begin{array}{c|c|c} A & B & T \\ \hline M & \Phi & \underline{0} \end{array}\right] = \left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}\right)$$

To remove singularity of $\Phi = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ one can exchange column 8 with 5. This corresponds to the following equivalent H with column ordering

$$\left(\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right)$$

$$= \left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right).$$

1, 2, 3, 4, 10, 6, 7, 5, 11, 12, 8, 9.

Then dividing codeword to $(\underline{s}, \underline{p}_1, \underline{p}_2)$ we have

$$\underline{s} \cdot A^T + \underline{p}_1 B^T + \underline{p}_2 T^T = \underline{0}$$

$$\underline{s} \cdot M^T + \underline{p}_1 \Phi^T = \underline{0}$$

where $A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$

$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$, $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

$M = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$, $\Phi = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$

Let's encode $(1,0,0,0,0,0) = \underline{s}$

$$s \cdot A^T = [1,0,0,0,0,0] \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\underline{s} \cdot A^T = [1 \ 1 \ 0 \ 1]$$

$$\underline{s} \cdot M^T = [0 \ 1]$$

$$s \cdot M^T + \underline{p}_1 \Phi^T = 0 \Rightarrow (\Phi^T)^{-1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\underline{p}_1 = s \cdot m^T (\phi^T)^{-1} = [0 \quad 1] \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\underline{p}_1 = [0 \quad 1]$$

$$\underline{s} \cdot A^T + \underline{p}_1 B^T + \underline{p}_2 T^T = \underline{0}$$

$$[1 \; 1 \; 0 \; 1] + [0 \quad 1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}^T + \underline{p}_2 T^T = \underline{0}$$

$$[1 \; 1 \; 0 \; 1] + [0 \; 1 \; 0 \; 0] + \underline{p}_2 T^T = 0$$

$$[1 \; 0 \; 0 \; 1] = p_2 T^T = [p_{2,1}, p_{2,2}, p_{2,3}, p_{2,4}] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow p_{2,1} = 1, \; p_{2,2} = 0, \; p_{2,3} = 1, \; p_{2,4} = 0.$$

So

$$\underline{v} = [1 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 1 \; 1 \; 0 \; 1 \; 0]$$

(from Shokrollahi's paper)

Decoding of LDPC Codes

Decoding of LDPC codes is performed <u>message passing</u> or <u>belief Propagation</u> (BP) algorithm. BP is an iterative algorithm where in each iteration message nodes send the likelihood of their value to all check nodes to which they are connected and check nodes send messages to variable (message) nodes based on what they have received from <u>other</u> message nodes. Message nodes and check nodes exclude what they have received from one another when they send a message.

In BP the message sent from a message node is based on that node's received information and what it gets from check nodes connected to it (except the one it wants to send the message to). These are in the form of probability or likelihood ratio.

In particular, a message node v sends to a check node c the probability (or likelihood) of v having a certain value given its observation and what it has received in the previous iteration from its neighboring check nodes <u>other than c</u>.

In the same way, the message c sends to v is the probability that c has a certain value given all the message passed to c in the previous iteration from message nodes <u>other than v</u>.

Likelihood ratio of a binary random variable x is

$$L(x) = \frac{P(x=0)}{P(x=1)}$$

and the condition likelihood ratio of x given y is

$$L(x|y) = \frac{P(x=0|y)}{P(x=1|y)}$$

if x is an equi-probable random variable

$$L(x|y) = L(y|x).$$

So, if $y_1, y_2, \ldots, y_d$ are independent random variables:

$$L(x) = \prod_{i=1}^{d} L(x|y_i)$$

or

$$\log L(x) = \sum_{i=1}^{d} \log L(x|y_i) \quad \left[\begin{array}{l} \log \text{ is based } e, \text{ i.e.} \\ \ln \end{array} \right.$$

Now assume $x_1, x_2, \ldots x_\ell$ are binary random variables and $y_1, y_2, \ldots y_\ell$ are random variables.

We would like to find

$$\ln L(x_1 \oplus x_2 \oplus \cdots \oplus x_\ell \mid y_1, \ldots, y_\ell)$$

Note that if we let

$$2P[x_1 = 0 \mid y_1] - 1 = p$$

and

$$2P[x_2 = 0 \mid y_2] - 1 = q$$

Then

$$P[x_1 \oplus x_2 = 0 \mid y_1, y_2] = P[x_1 = 0, x_2 = 0 \mid y_1, y_2] + P[x_1 = 1, x_2 = 1 \mid y_1, y_2]$$

$$= P[x_1 = 0 \mid y_1] P[x_2 = 0 \mid y_2] + P[x_1 = 1 \mid y_1] P[x_2 = 1 \mid y_2]$$

$$= \frac{1+p}{2} \cdot \frac{1+q}{2} + \frac{1-p}{2} \cdot \frac{1-q}{2} = \frac{2+2pq}{2} = 1 + pq$$

So

$$2P[x_1 \oplus x_2 = 0 \mid y_1, y_2] - 1 = pq$$

Therefore,

$$2P[x_1 \oplus x_2 \oplus \cdots \oplus x_\ell \mid y_1, y_2, \ldots, y_\ell] - 1 = \prod_{i=1}^{\ell} [2P(x_i = 0 \mid y_i) - 1]$$

let $\lambda_i = \log_e \frac{P(x_i = 0 \mid y_i)}{P(x_i = 1 \mid y_i)}$ be the log-likelihood ratio of x_i given y_i.

So,

$$P(x_i=0|y_i) = \frac{e^{\lambda_i}}{e^{\lambda_i}+1}$$

$$2P(x_i=0|y_i)-1 = \frac{e^{\lambda_i}-1}{e^{\lambda_i}+1} = \frac{e^{\lambda_i/2}-e^{-\lambda_i/2}}{e^{\lambda_i/2}+e^{\lambda_i/2}}$$

or

$$2P(x_i=0|y_i) = \tanh\left(\frac{\lambda_i}{2}\right)$$

Finally,

$$\ln L[x_1\oplus\cdots\oplus x_\ell|y_1,\ldots,y_\ell] = \ln \frac{1+\prod_{i=1}^{\ell}\tanh(\frac{\lambda_i}{2})}{1-\prod_{i=1}^{\ell}\tanh(\frac{\lambda_i}{2})}$$

Let $m_{vc}^{(\ell)}$ be the message passed from message node v to check node c in iteration ℓ. Similarly, denote by $m_{cv}^{(\ell)}$ the message from c to v. Then, the update equations in BP are

$$m_{vc}^{(\ell)} = \begin{cases} m_v, & \text{if } \ell = 0, \\ m_v + \sum_{c' \in C_v \setminus \{c\}} m_{c'v}^{(\ell-1)}, & \text{if } \ell \geq 1, \end{cases}$$

$$m_{cv}^{(\ell)} = \ln \frac{1+\prod_{v' \in V_c \setminus \{v\}} \tanh\left(m_{v'c}^{(\ell)}/2\right)}{1-\prod_{v' \in V_c \setminus \{v\}} \tanh\left(m_{v'c}^{(\ell)}/2\right)},$$

where C_v is the set of check nodes connected to v.
Similarly V_c are variable nodes connected to c.

Bit-flip decoding algorithm:

This method was devised by Gallager. When we compute syndromes (the value of check nodes), if they are all zero then we assume there is no error and stop.

Then we find for each variable node, the number of failed (1) check nodes and flip the one with maximum number of failed check nodes connected to it.

We then re-calculate the syndromes and flip the bit that is most connected to those with value 1.

We continue iteration above until either all check nodes have zero value or until a certain number of pre-determined iterations have been done with no success (failure in this case). This simple BP algorithm is given below:

1- Compute syndromes by $\underline{r} \cdot \underline{H}^T = \underline{s}$. If all check-sums are 0 stop.

2- Find the number of failed parity check equations for each node.

Denod the number of failed check node for each message nod by f_i, $i = 1, 2, \ldots, n$.

Step 3: Identify the set of bits S for which f_i is the largest.

Step 4: Flip bits in S.

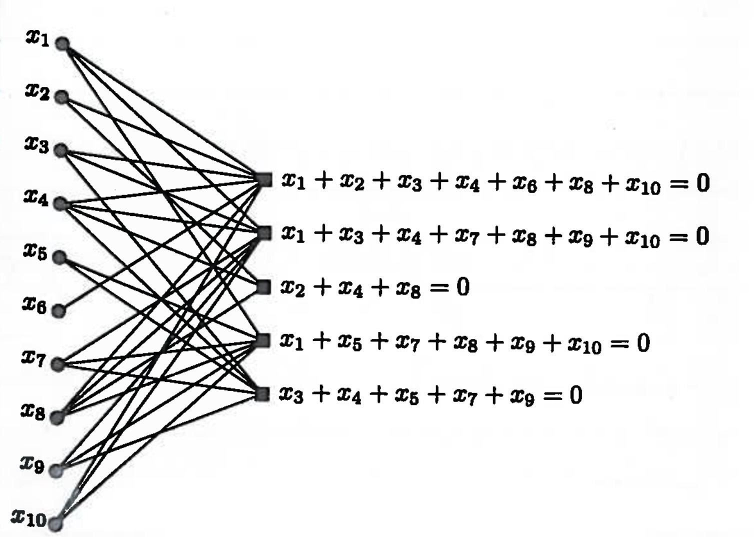
Step 5: Repeat steps 1 to 4 until the parity-check sums are zero (success). Or a preset maximum number of iterations is reached (decoding failure)

Example

Assume that we have used this Code and have received

0 0 0 0 0 0 0 1 0 0

that is $x_8 = 1$ and $x_i = 0 \quad i \neq 8$



Step 1: Compute Syndromes:

We have:

$$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\} = \{0,0,0,0,0,0,0,1,0,0\}$$

This results in syndromes as:

$$C = \{c_1, c_2, c_3, c_4, c_5\} = \{1,1,1,1,0\}$$

Obviously, this indicates an error.

Step 2: Find the number of failed parity check equations for each node:

Below table shows the frequency of occurrence of each node in the failed parity check equations:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
f_i	3	2	2	3	1	1	2	4	2	3

Step 3: Identify the bits for which the frequency of occurrence is the largest.

From Step 2, this is clearly x_8, which has occurred in all four failed parity check equations.

Step 4: Flip bits from Step 3.

By flipping x_8, the code word will be $X = \{0,0,0,0,0,0,0,0,0,0\}$.

This results in all zero syndromes, which means successful LDPC decoding.