Week 7 : Session 1

LOW DENSITY PARITY CHECK CODES : AN INTRODUCTION

Outline :

- Introduction
- Tanner Graphs (Graphically represent the Parity check Matrix of the LDPC code)
- Two types of LDPC codes
  - Regular LDPC codes
  - Irregular LDPC codes
- Construction of regular LDPC codes
  - Gallager's construction
  - MacKay's construction
- Random construction of irregular LDPC codes.

Definitions :

⊙ The density of a source of random bits is the expected fraction of 1 bits.

Eg: Source, $v = [0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1]$

Out of 10 bits, the No. of 1's are 4.

So, density $= \frac{4}{10} = 0.4$.

⊙ A source is sparse if its density is less than 0.5.

⊙ A vector v is very sparse if its density vanishes as its length increases.

⊙ The overlap between two vectors is the No. of 1's in common between them.

Eg: $v_1 = [0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1]$

$v_2 = [0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1]$

So, overlap = 3.

LDPC codes :

⊙ Low Density Parity Check (LDPC) codes are Error Correcting codes specified by a parity check matrix H containing mostly 0's and only a small number of 1's.

⊙ We say a LDPC code is Regular LDPC code, if the No. of 1's in each row and the No. of 1's in each column are same. So, all the rows of parity check matrix and all the columns of parity check matrix have the same No. of 1's.

Eg. $$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Here, the No. of 1's in each row is 4.
and the No. of 1's in each column is 1.

And clearly, the No. of 1's in each row will be greater than the No. of 1's in each column.

⊙ A regular (n, w_c, w_r) LDPC code is a code of block length n with a $m \times n$ parity check matrix, where each column contains a small fixed number $w_c \geq 3$ of 1's and each row contains a small fixed number $w_r \geq w_c$ of 1's.

No. of 1's columnwise $= n\, w_c$
No. of 1's row wise $= (n-k)\, w_r$

$$n\, w_c = (n-k)\, w_r.$$

Now, we want the No. of 1's in each column should be atleast 3. This is because the performance of LDPC code is good if the column weight is atleast 3.

A regular LDPC code is defined by a low density parity check matrix H,

$$H = \begin{bmatrix} \quad \end{bmatrix}_{m \times n}, \text{ with } m \geq n-k$$

So, the block length is n. The No. of 1's in each column

is given by w_c which is atleast 3, NO. of 1's in each row is given by w_r. w_c is same for all the columns, w_r is same for all the rows.

⊙ In other words,

* Each parity check constraint involves w_r codebits, and each codebit is involved in w_c constraints.
* Low density implies that, $w_c << n$ and $w_r << n$.
* NO. of 1's in the parity check matrix $H = w_c \cdot n$
  $= w_r \cdot m$
* $m \geq n-k$

$$\Rightarrow R = \frac{k}{n} \geq 1 - \left(\frac{w_c}{w_r}\right),$$

and thus $w_c < w_r$.

Regular low-density parity check code:

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

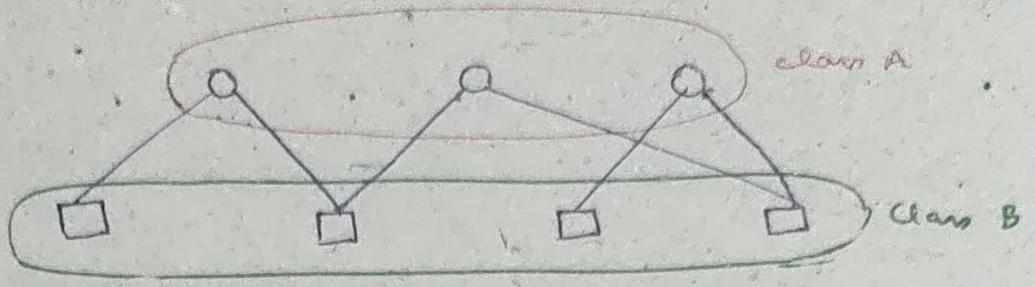
Example of a regular low density code matrix;
$n = 20$, $w_c = 3$ and $w_r = 4$. ——— Fig 1

$$R = 1 - \frac{w_c}{w_r} = 1 - \frac{3}{4} = \frac{1}{4}.$$

Tanner Graphs.

⊙ A bipartite graph is one in which the nodes can be partitioned into two classes, and no edge can connect nodes from the same class.

Eg.



⊙ A Tanner graph for an LDPC code is an bipartite graph such that:

- One class of nodes is the "variable nodes" corresponding to n bits in the codeword.
- Second class of nodes is "check nodes" corresponding to m parity check equations.
- An edge connects a variable node to the check node if and only if that particular bit is included in the parity check equation.

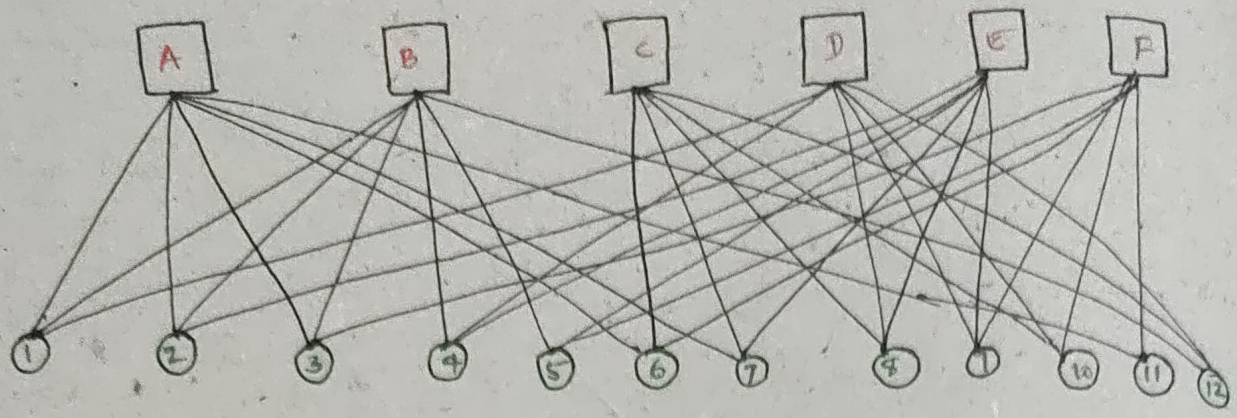
⊙ Example of a regular low density code matrix;

$n = 12$, $w_c = 3$, $w_r = 6$.

12 code bits

$$H = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} \leftarrow A \\ \leftarrow B \\ \leftarrow C \\ \leftarrow D \\ \leftarrow E \\ \leftarrow F \end{matrix}$$

6 parity check equations.

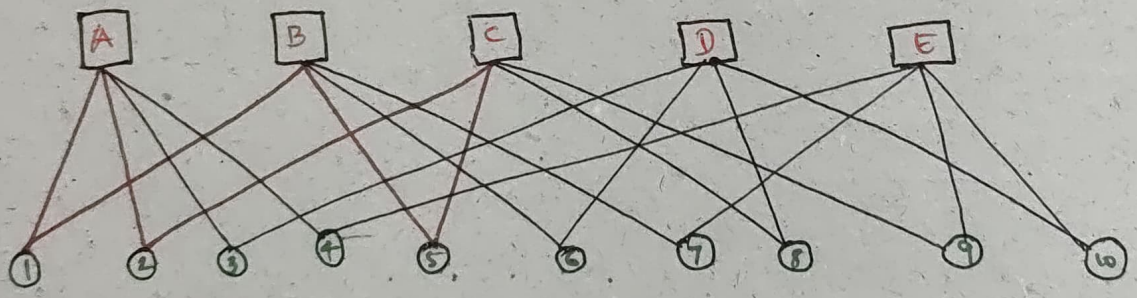


- Corresponding to the 6 parity check equations, there'll be 6 parity check nodes.
- Corresponding to the 12 code bits, there'll be 12 variable nodes.

CYCLE :

A cycle of length l in a Tanner graph is a path comprised of l edges from a node back to the same node.

Example: The bipartite graph has a cycle of length 6.



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

GIRTH :

The length of the smallest cycle in the graph is known as its Girth.

When decoding LDPC codes using Sum-product algorithm, the number of independent iterations of the algorithm is proportional to the Girth of its associated Tanner graph.

The Girth of the above Tanner graph is 6.

Gallager's Construction for regular (n, w_c, w_r) code.

⊙ Let n → Transmitted block length of an information sequence of length k.

m → No. of parity check equations

⊙ Construct a $m \times n$ matrix with w_c 1's per column and w_r 1's per row

An (n, w_c, w_r) code.

⊙ Divide a $m \times n$ matrix into w_c $(m/w_c \times n)$ sub-matrices, each containing a single 1 in each column.

⊙ The first of these sub-matrices contains all 1's in descending order. (ii) the i^{th} row contains 1's in columns $(i-1) \cdot w_r + 1$ to $i \cdot w_r$.

⊙ The other sub-matrices are merely column permutations of the first sub matrix.

Example of a regular low density code matrix; $n = 20$, $w_c = 3$, $w_r = 4$ is shown in figure 1.

MacKay's construction

- An m by n matrix (m rows, n columns) is created at random with weight per column w_c, and weight per row w_r, and overlap between any two columns no greater than 1.
- Another way of constructing regular LDPC codes is to build the parity check matrix from non-overlapping random permutation matrices. Reference:

"Good Error-Correcting Codes Based on Very Sparse Matrices" by David J.C. MacKay in IEEE Transactions on Information Theory, pp. 399-431, Mar. 1999.

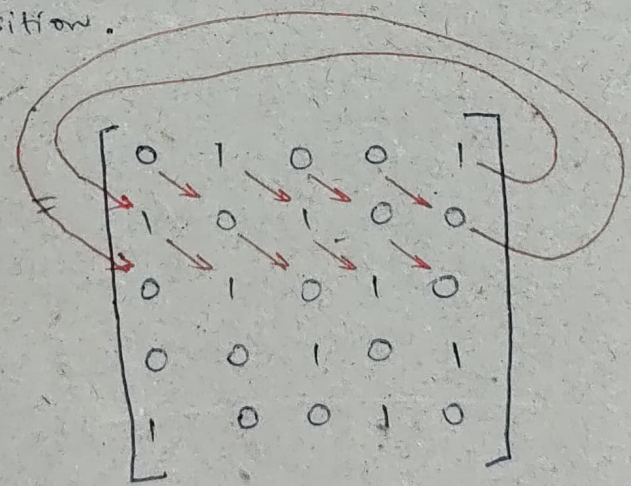
Construction of low density parity check codes

- A permutation matrix is just the identity matrix with its row reordered.

Eg. $P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$

- A circulant matrix is defined by the property that each row is a cyclic shift of the previous row to the right by one position.

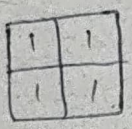
Eg.

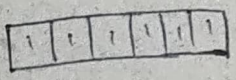


MacKay's construction

 → denotes a Square Matrix which has
- Singer 1's in each column
- singer 1's in each row

 → denotes a matrix where
- No. of rows is n
- No. of columns is 2^n
- Each row will have two 1's but
- Each column will have singer 1.

 → denotes a square matrix where
- No. of rows is n
- No. of columns is also n
- Each row will have two 1's and
- Each column will also have two 1's.

 → denotes a matrix where
- No. of rows is n
- No. of columns is 6n
- Each row will have six 1's
- Each column will have singer 1.

 → denotes parity check matrix where
- Each column will have three 1's
- Each row will have six 1's

This helps in generating $R = \frac{1}{2}$ LDPC code.

Schematic illustration of Regular Gallager codes.

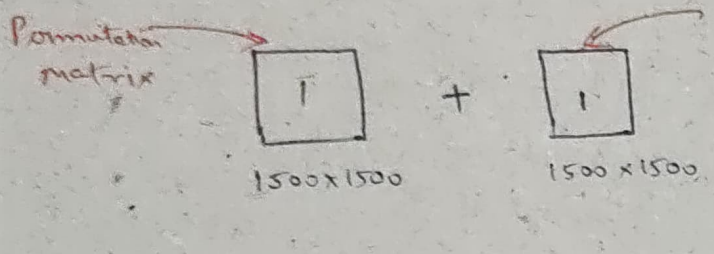
Let's say, the No. of coded bits, $n = 3000$

As Rate, $R = \frac{1}{2}$, then $n - k = 1500$

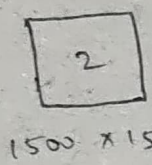
So, we design 500 × 500 permutation matrix. Each one of them is a 500 × 500 permutation matrix. If we stack them all together, we get a matrix which has 1500 rows and 3000 columns.

So, this will result in a parity check matrix which has 1500 rows and 3000 columns. Each column of this matrix will have three 1's and each row of this matrix will have six 1's.

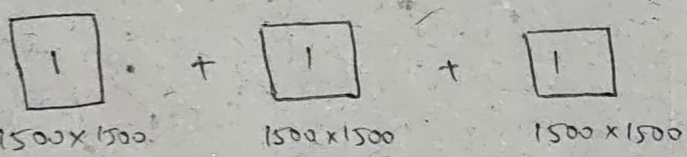
Another way to generate this parity check matrix is using Permutation matrices which are added together. (ii) we take two 1500 x 1500 permutation matrices



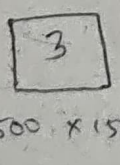
If we add two such permutation matrices, we get a matrix of dimension 1500 x 1500 where each row and each column has 2 1's.



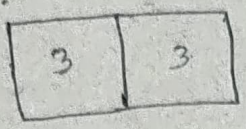
Now, lets take three such permutation matrices and add them.



Remember, when we add these permutation matrices, we've to ensure that the 1's are at different location. (ii) we want non-overlapping 1's. So, if we add three such permutation matrices, we'll get a permutation matrix of dimension 1500 x 1500 where each column is of weight 3 and each row is of weight 3.



Now, if we take two such permutation matrices and stack them up as shown below, we get a matrix of dimension 1500 x 3000 where each column has weight 3, and each row has weight 3+3 = 6.



Schematic Illustration of Regular Gallager codes.

This is the another way of constructing a parity check matrix whose column weight is 3 and row weight is 6.

Notation: An integer represents a number of permutation matrices superposed on the surrounding square.

Column Weight	Fraction of columns	Row weight	Fraction
3	1	6	1

Irregular LDPC codes

In Regular LDPC code, each row has same number of 1's and each column has same number of 1's. Now we are relaxing that condition. (ii) Now we are saying, each coded bit can participate in different number of parity check equations. Say for example, there are some code bits which are participating in 3 parity check equations. Similarly, there are some code bits which are participating in 4 parity check equations. Similarly in each parity check equation there are different number of bits participating. In some, there are 6, or there are 7. So, what do we gain by allowing this flexibility?

Remember, if more number of bits participating in a parity check equation, it is lot difficult to satisfy that parity constraint, but it is a very powerful constraint. However, if there are lesser number of bits participating in a parity check equation, then its easy to satisfy, but it may not be very powerful.

So, we are kind of combining ease of computation or ease of decoding: If there are smaller number of bits participating parity check equations, those parity check constraint can be easily satisfied. So, maybe when we talk about iterative decoding algorithm, those parity check constraints can be satisfied and then they can propagate information across the graph. Whereas by keeping some weak parity constraints, and some strong parity constraint, overall code will work well.

So, by allowing this flexibility of having different number of bits participating in a parity check equation and different number of bits participating in a parity check constraint, we are kind of making it more general.

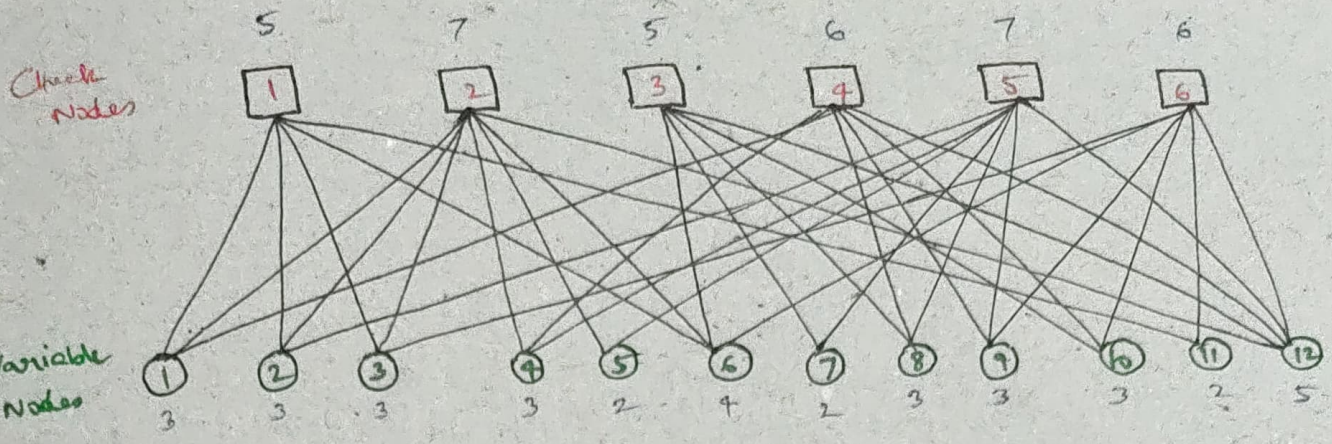
So, we define the number of code bits participating in a parity check constraint and each code bit participating in how many equation, we show them by some distribution.

- For an irregular LDPC code, the degrees of each set of nodes are chosen according to some distribution.
- A degree distribution $\gamma(x) = \sum_i \gamma_i x^{i-1}$ is simply a polynomial with non negative real coefficients satisfying $\gamma(1) = 1$.
- An irregular low-density code is a code of block-length N with a sparse parity check matrix where column distribut $\lambda(x)$ and row distribution $\rho(x)$ is respectively given by

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

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

where λ_i and ρ_i denote the fraction of edges incident to variable and check nodes with degree i, respectively.



The corresponding parity check matrix, $H = \begin{bmatrix} & \end{bmatrix}_{6\times 12}$.

⊙ We have,

5	Code bits	connected to	check Node 1
7	Code bits	connected to	check Node 2
5	Code bits	connected to	check Node 3
6	Code bits	connected to	check Node 4
7	Code bits	connected to	check Node 5
6	Code bits	connected to	check Node 6

We have Rows of weights 5, 7, 5, 6, 7, 6.

Out of these, $\frac{2}{6}$ has weight 5.

(i) $\frac{1}{3}^{rd}$ of the Rows have weight 5 (ii) Five 1's.

(iii) $\frac{1}{3}^{rd}$ of the Rows have weight 6 (ii) Six 1's

and $\frac{1}{3}^{rd}$ of the Rows have weight 7 (i) Seven 1's.

Row distribution, $\rho(x) = \frac{1}{3}x^4 + \frac{1}{3}x^5 + \frac{1}{3}x^6$

⊙ We have;

3 parity check constraints connected to Variable Node 1.

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

We have parity check constraints 3, 3, 3, 3, 2, 4, 2, 3, 3, 3, 2, 5

Out of these 12, we have 3 variable nodes connected to parity check constraint 2. (ii) $\frac{3}{12}$.

(ii) $\frac{1}{4}^{th}$ of variable nodes are connected to parity check constraint 2

iii) $\frac{7}{12}^{th}$ of variable nodes are connected to parity check constraints 3

$\frac{1}{12}^{th}$ of variable nodes are connected to parity check constraint 4

$\frac{1}{12}^{th}$ of variable nodes are connected to parity check constraint 5.

Column distribution, $\lambda(x) = \frac{1}{4}x + \frac{7}{12}x^2 + \frac{1}{12}x^3 + \frac{1}{12}x^4$

So, this is how we write the degree distribution of a irregular LDPC code.

Repeat:

An Irregular LDPC code can be described by Column and Row degree distribution, which tells us, what fraction of nodes are connected to how many parity check equation, and what fraction of code bits are participating in a particular parity check equation.

Construction of irregular LDPC code

Step 1: Selecting a profile that describes the desired number of columns of each weight and the desired number of rows of each weight.

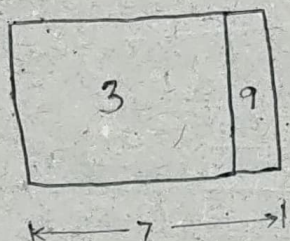
Step 2: Construction method; (ii) algorithm for putting edges between the vertices in a way that satisfies the constraints.

Random construction of Irregular LDPC codes

The edges are placed "completely at random" subject to the profile constraints. One way of implementing it is shown below.

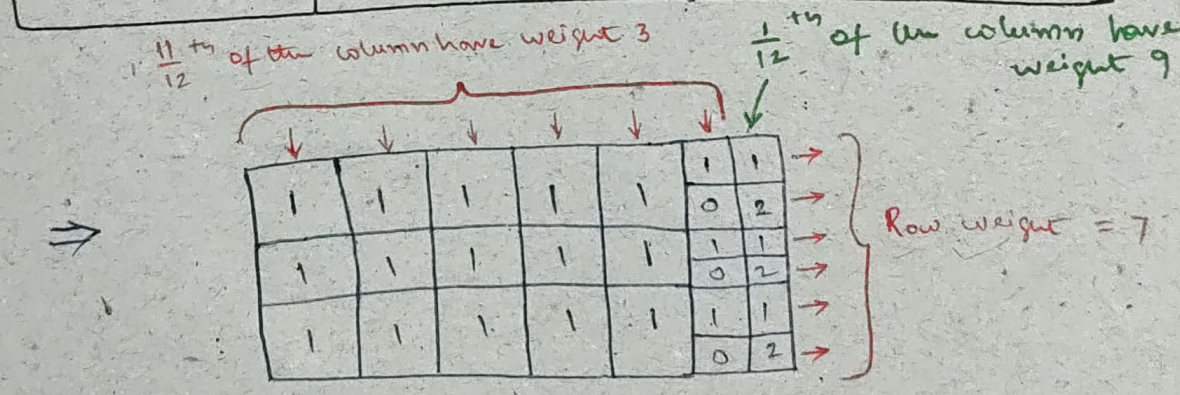
- Make a list of all columns in the matrix, with each column appearing in the list a number of times equal to its weight.
- Make a similar list of all rows in the matrix, with each row appearing in the list a number of times equal to its weight.
- Map one list onto the other by a random permutation, taking care not to create duplicate entries.

Construction of Irregular LDPC codes



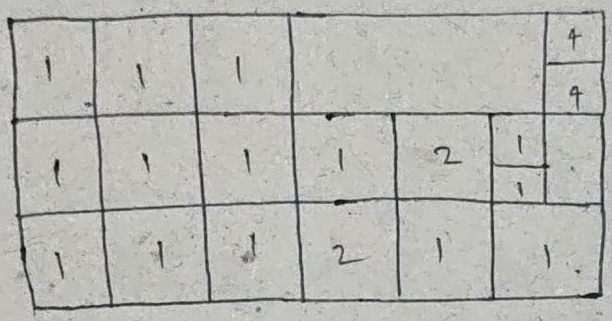
Notation: integers "3" and "9" represent the column weights.

Column Weight	Fraction of Columns	Row weight	Fraction
3	11/12	7	1
9	1/12		

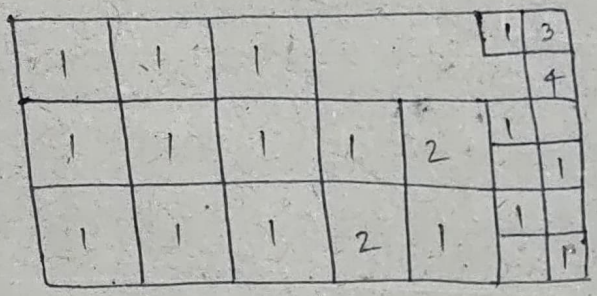


Notation: An integer represents a number of permutation matrices superposed on the surrounding square. Horizontal and Vertical lines indicate the boundaries of the permutation blocks.

Another Example:



Another Example:



We see that there are multiple ways (using permutation matrices) to construct the LDPC code. When we are talking about constructing LDPC codes, we are trying to construct the low density parity check matrix, which has certain profile:

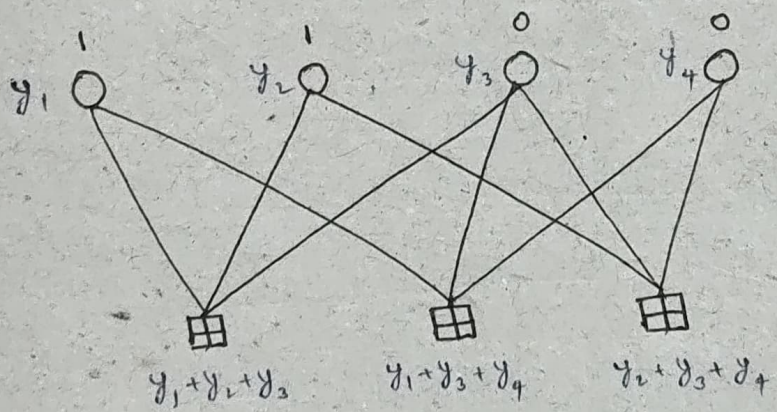
Week 7 : Session 2

DECODING OF LDPC CODES - 1 : Bit Flipping Algorithm.

In this section, we'll talk about simple Bit Flipping Algorithm. In the next section, we'll talk about Probabilistic decoding.

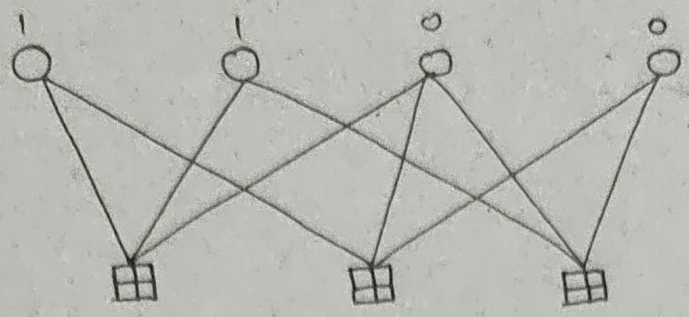
- Message passing algorithm used for decoding LDPC codes is an iterative decoding algorithm that uses the structure of Tanner graph.
- In each iteration of the algorithm
  - Each variable node sends a message to each check node it is connected to ;
  - Each check node sends a message to each variable node it is connected to ;
  - For each code symbol, we compute the aposteriori probability that the symbol takes on the value "1", given all the received symbols and that all the parity check equations are satisfied.

Received codeword



Replace y_1 with 1
y_2 with 1
y_3 with 0
y_4 with 0

Received Codeword



1+1+0 = 0

1+0+0 = 1

1+0+0 = 1

Parity check constraint is satisfied.

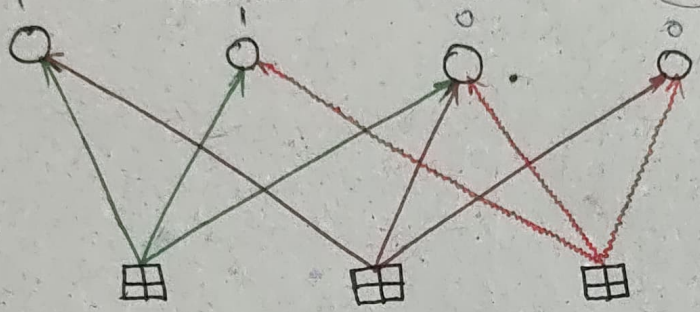
Parity check constraint is not satisfied

Dilemma — Stay, Flip

Dilemma — Stay, Flip

Dilemma — Stay, Flip, Flip

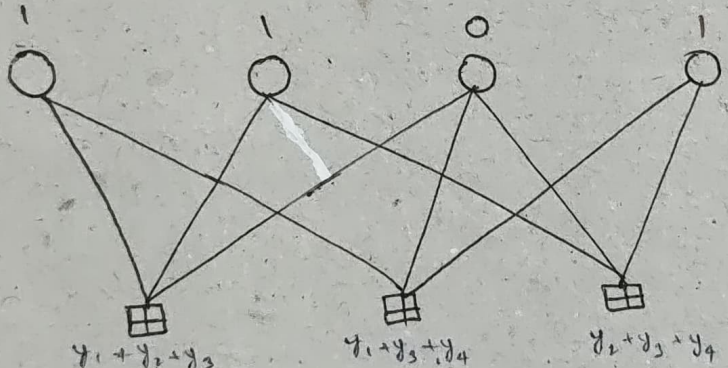
No dilemma. Have to Flip. — Flip, Flip



1+1+0 = 0

1+0+0 = 1

1+0+0 = 1



$y_1 + y_2 + y_3$

$y_1 + y_3 + y_4$

$y_2 + y_3 + y_4$

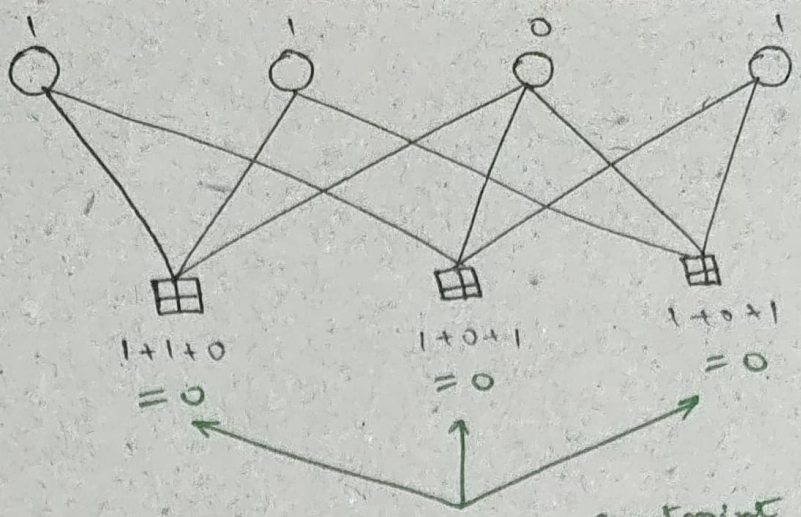
Now, repeat the process again.

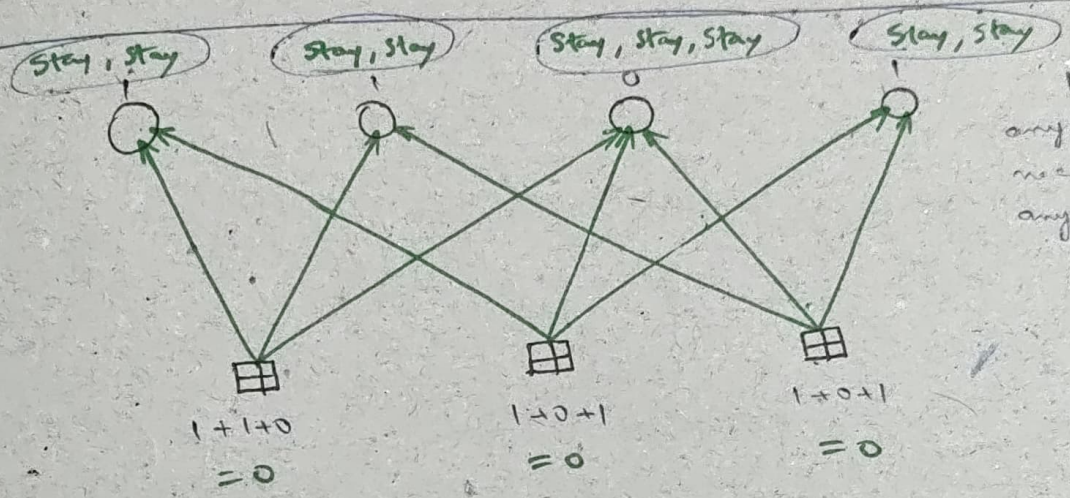
Replace y_1 with 1

1 y_2 with 1

y_3 with 0

y_4 with 1





No dilemmas anywhere. No need to flip any further.

Thus, the decoded sequence is 1101.

Week 7 : Session 3.

DECODING OF LDPC CODES - 2 : Belief Propagation Algorithm

In the last section, we talked about, how a simple bit flipping algorithm can be used, to correct errors in LDPC codes. We exploited the Tanner graph of the LDPC code to do local computation and then we pass that information to check Node, to check whether the parity check constraints are satisfied. Now; we are going to generalize this algorithm and we are going to talk about Probabilistic decoding of LDPC codes.

Theorem :

Consider a sequence of m independent random variables $A = [A_1, A_2, \ldots, A_m]$, where A_k is 0 or 1, A_2 is 0/1, A_m is 0/1, and $P(A_k = 1) = p_k$, $P(A_k = 0) = 1 - p_k$.

Then,

$$P(A \text{ has even parity}) = \frac{1}{2} + \frac{1}{2} \prod_{k=1}^{m} (1 - 2p_k)$$

and

$$P(A \text{ has odd parity}) = \frac{1}{2} - \frac{1}{2} \prod_{k=1}^{m} (1 - 2p_k).$$

Proof :

- Consider the function $\prod_{i=1}^{m} (1 - p_i + p_i t)$
- The coefficient of t^i is the probability of t i's.
- The function $\prod_{i=1}^{m} (1 - p_i - p_i t)$ is identical except for the fact that odd powers of t are negative.
- Adding these two functions, all even powers of t double up and odd powers cancel each other.

⊙ Letting $t=1$, and dividing by 2 we get the probability of getting even ones.

$$P(\text{A has even parity}) = \frac{1}{2} + \frac{1}{2}\prod_{k=1}^{m}(1-2p_k)$$

⊙ Similarly, we can prove

$$P(\text{A has odd parity}) = \frac{1}{2} - \frac{1}{2}\prod_{k=1}^{m}(1-2p_k).$$

Notation:

⊙ Consider the code with parity check matrix, H:

$$H = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Bit numbers 1, 2, 3 are taking part in the First parity check equation
Bit numbers 4, 5, 6 are taking part in the Second parity check equation
Bit numbers 1, 4, 7 are taking part in the third parity check equation
Bit numbers 2, 5, 8 are taking part in the fourth parity check equation

⊙ $c = [c_0, c_1, \ldots, c_{n-1}]$ is the codeword under consideration.

⊙ $X_i = (-1)^{c_i} \in \{+1, -1\}$, the BPSK modulated version of c_i.

(ii) $c_i = 0 \longrightarrow$ mapped to $+1$
$c_i = 1 \longrightarrow$ Mapped to -1

⊙ Assuming transmission over AWGN,

$$Y_i = X_i + n_i,$$

where $n_i \rightarrow$ Zero mean Gaussian with Variance σ^2.

⊙ $R_j = \{i : h_{j,i} = 1\}$

= location of 1's in row j of H

= the indices of the bits checked by the j^{th} parity check.

(ii) $R_1 = \{1, 2, 3\}$

$R_2 = \{4, 5, 6\}$

$R_3 = \{1, 4, 7\}$

$R_4 = \{2, 5, 8\}$

⊙ $C_i = \{j : h_{j,i} = 1\}$

= location of 1's in column i of H

= the parity checks involving the i^{th} codebit.

$j-i$

⊙ $R_{j \setminus i} = R_j \setminus \{i\}$

$i-j$

⊙ $C_{i \setminus j} = C_i \setminus \{j\}$

⊙ $c_{k,j}(i)$ = k^{th} bit in the j^{th} parity check involving the codebit c_i. (so $j \in C_i$ and $k \in R_j$).

⊙ $Y_{k,j}(i) = (-1)^{c_{k,j}(i)} + n_{k,j}(i)$, received signal corresponding to $c_{k,j}(i)$.

⊙ $p_i = P(c_i = 1 \mid Y_i = y_i)$

$= P(x_i = -1 \mid Y_i = y_i)$

$= \dfrac{1}{1 + \exp(2y_i/\sigma^2)}$

⊙ $P_{k,j}(i) = P\left(c_{k,j}(i) = 1 \mid Y_{k,j}(i)\right)$.

Theorem :

The aposteriori probability (APP) ratio for c_i given the received word $y = [y_0, y_1, \ldots, y_{n-1}]$ and given the event $S_i = \{$ the bits in c satisfy the parity check constraints involving $c_i \}$, is given by

$$\frac{P(c_i = 0 \mid y, S_i)}{P(c_i = 1 \mid y, S_i)} = \frac{(1-P_i)}{P_i} \frac{\prod_{j \in C_i} \left(1 + \prod_{i' \in R_{j \setminus i}} (1 - 2P_{i',j}(i))\right)}{\prod_{j \in C_i} \left(1 - \prod_{i' \in R_{j \setminus i}} (1 - 2P_{i',j}(i))\right)}$$

Proof :

⊙ From Bayes's Rule,

$$\frac{P(c_i = 0 \mid y, S_i)}{P(c_i = 1 \mid y, S_i)} = \frac{\overbrace{P(c_i = 0 \mid y_i)}^{1-P_i} \; P(S_i \mid c_i = 0, y)}{\underbrace{P(c_i = 1 \mid y_i)}_{P_i} \; P(S_i \mid c_i = 1, y)} \quad \text{— (1)}$$

⊙ Lets consider the term $P(S_i \mid c_i = 0, y)$.

Given $c_i = 0$, S_i holds if each of w_c parity checks involving c_i has the property that the $w_r - 1$ bits in the check other than c_i have even parity.

⊙ For parity check $j \in C_i$, the probability that the $w_r - 1$ bits other than c_i have even parity is given by the lemma to be :

$$\frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2P_{i',j}(i))$$

⊙ The independence of the y_i's mean that the probability that all w_c parity checks involving c_i are satisfied (given $c_i = 0$) is just

$$P(S_i \mid c_i = 0, y) = \prod_{j \in C_i} \left(\frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2P_{i',j}(i))\right) \quad \text{— (2)}$$

⊙ Similar analysis assuming $c_i = 1$ yields

$$P(s_i \mid c_i = 1, y) = \prod_{j \in C_i} \left(\frac{1}{2} - \frac{1}{2} \prod_{i' \in R_j \setminus i} (1 - 2P_{i'j}(i)) \right)$$

③

⊙ Substitute ② and ③ in ① would result to proof.

Now, lets look at the quantities that we are computing here.

⊙ $r_{j,i}(x)$ → Message passed from the j^{th} check node to the bit node $X_i = x$.

$$r_{j,i}(+1) = P(\text{Parity check } j \text{ satisfied} \mid c_i = 0, \text{ other bits in check } j \text{ have distributions given by } q)$$

$$= \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_j \setminus i} (1 - 2q_{i'j}(-1))$$

and so

$$r_{j,i}(-1) = P(\text{Parity check } j \text{ satisfied} \mid c_i = 1, \text{ other bits in check } j \text{ have distributions given by } q)$$

$$= P(\text{Parity check } j \text{ not satisfied} \mid c_i = 0, \text{ other bits in check } j \text{ have distributions given by } q)$$

$$= 1 - r_{j,i}(+1).$$

⊙ $q_{i,j}(x)$ → Message passed from the bit node $X_i = x$ to the j^{th} check node.

$$q_{i,j}(+1) = P(X_i = +1 \mid y_i, \text{ information from check nodes other than } j^{th} \text{ check node})$$

$$\frac{q_{i,j}(+1)}{q_{i,j}(-1)} = \frac{(1 - p_i)}{p_i} \cdot \frac{\prod_{j' \in C_{i,j}} r_{j',i}(+1)}{\prod_{j' \in C_{i,j}} r_{j',i}(-1)}$$

For all i,j such that $h_{j,i} = 1$. (So i indexes the bit nodes and j indexes the parity checks).

Step 0: Initialize

- Set $p_i = P(c_i = 1 \mid Y_i = y_i) = \frac{1}{\exp(2y_i/\sigma^2)}$.
- $q_{i,j}(+1) = 1 - p_i$
- $q_{i,j}(-1) = p_i$

Step 1: Pass information from check nodes to bit nodes

- $r_{j,i}(+1) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j,i}} \left(1 - 2 q_{i',j}(-1)\right)$
- $r_{j,i}(-1) = 1 - r_{j,i}(+1)$

Step 2: Pass information from bit nodes to check nodes.

- $q_{i,j}(+1) = K_{i,j}(1-p_i) \prod_{j' \in C_{i,j}} r_{j',i}(+1)$
- $q_{i,j}(-1) = K_{i,j}(p_i) \prod_{j' \in C_{i,j}} r_{j',i}(-1)$
- Here, the constants $K_{i,j}$ are chosen so as to guarantee that $q_{i,j}(+1) + q_{i,j}(-1) = 1$.

Step 3: Compute the APP ratios for each bit position i.

- $Q_i(+1) = K_i(1-p_i) \prod_{j \in C_i} r_{j,i}(+1)$
- $Q_i(-1) = K_i p_i \prod_{j \in C_i} r_{j,i}(-1)$
- Here, the constants K_i are chosen so as to guarantee that $Q_i(+1) + Q_i(-1) = 1$.

Step 4 : Compute the hard decisions and decide if it's time to stop.

$$\hat{c}_i = \begin{cases} 1 & \text{if } Q_i(-1) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

If $\left([\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{n-1}] H^T = 0 \right)$ (or)

Maximum No. of iterations reached,

then Stop, else repeat Steps 1-4.

Example :

c_0	c_1	c_2
c_3	c_4	c_5
c_6	c_7	

We have two horizontal parity check constraints.

$$c_0 + c_1 + c_2 = 0 \quad \text{---} \; (1)$$

$$c_3 + c_4 + c_5 = 0 \quad \text{---} \; (2)$$

We have two vertical parity check constraints.

$$c_0 + c_3 + c_6 = 0 \quad \text{---} \; (3)$$

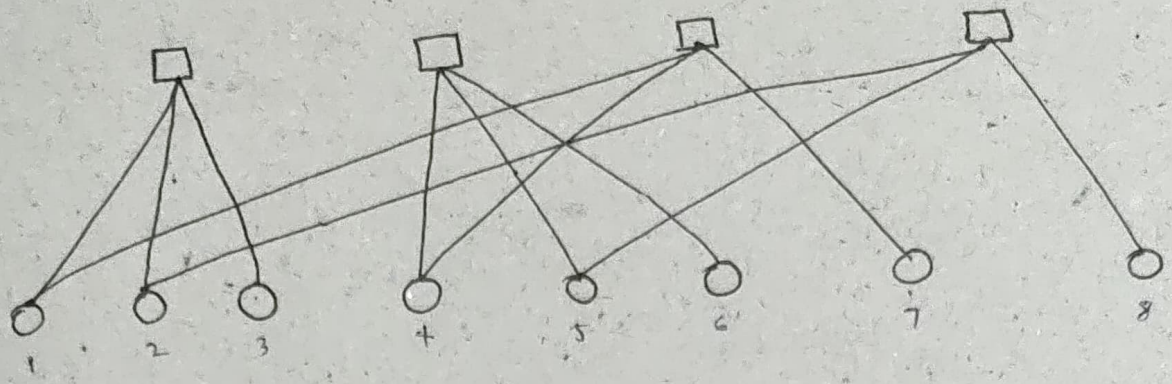
$$c_1 + c_4 + c_7 = 0 \quad \text{---} \; (4)$$

⊙ Consider the code with parity check matrix, H

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow (1) \\ \leftarrow (2) \\ \leftarrow (3) \\ \leftarrow (4) \end{matrix}$$

⊙ $n = 8$; $m = n - k = 4$; $d_{min} = 3$.

⊙ For this H matrix, we can draw Tanner graph.



1	0	1
0	1	1
1	1	

modulation →

-1	+1	-1
+1	-1	-1
-1	-1	

↓ transmission (AWGN)

$\sigma^2 = 0.5$

0 +0.2 +1	0 +0.2 +1	1 -0.9 -1
0 +0.6 +1	0 +0.5 +1	1 -1.1 -1
1 -0.4 -1	1 -1.2 -1	

Hard decision:

Received info. > 0 → +1 was sent

Received info < 0 → -1 was sent

Mapping

0 → +1

1 → -1

⇒

1	0	1
0	1	1
1	1	

Sent

0	0	1
0	0	1
1	1	

Received

2 bits are in error.

Now, lets apply belief propagation algorithm.

Initialization :

- $q_{i,j}(x) = \dfrac{1}{1 + \exp(-2xy_i/\sigma^2)}$

  for each i, j such that $h_{j,i} = 1$.

- $q_{0,0}(-1) = q_{0,2}(-1) = 0.310$ and

  $q_{0,0}(+1) = q_{0,2}(-1) = 0.690$

- $q_{1,0}(-1) = q_{1,3}(-1) = 0.310$ and

  $q_{1,0}(+1) = q_{1,3}(+1) = 0.690$

- $q_{2,0}(-1) = 0.973$ and

  $q_{2,0}(+1) = 0.027$

- $q_{3,1}(-1) = q_{3,2}(-1) = 0.083$ and

  $q_{3,1}(+1) = q_{3,1}(+1) = 0.917$.

- $q_{4,1}(-1) = q_{4,3}(-1) = 0.119$ and

  $q_{4,1}(+1) = q_{4,3}(+1) = 0.881$

- $q_{5,1}(-1) = 0.988$ and

  $q_{5,1}(+1) = 0.012$

- $q_{6,2}(-1) = 0.832$ and

  $q_{6,2}(+1) = 0.168$

⊙ $q_{7,3}(-1) = 0.992$ and

$q_{7,3}(+1) = 0.008$

Now compute $r_{j,i}$'s from $q_{i,j}$'s :

$$r_{0,0}(+1) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{0,0}} (1 - 2 q_{i',0}(-1))$$

$$= \frac{1}{2} + \frac{1}{2} (1 - 2 q_{1,0}(-1))(1 - 2 q_{2,0}(-1))$$

$$= \frac{1}{2} + \frac{1}{2} (1 - 2(0.31))(1 - 2(0.973))$$

$$= 0.320.$$

In similar way,

$r_{0,1}(+1) = 0.5 + 0.5 (1 - 2(0.31))(1 - 2(0.973)) = 0.32$

$r_{0,2}(+1) = 0.5 + 0.5 (1 - 2(0.31))(1 - 2(0.31)) = 0.57.$

$r_{1,3}(+1) = 0.5 + 0.5 (1 - 2(0.119))(1 - 2(0.988)) = 0.128$

$r_{2,0}(+1) = 0.5 + 0.5 (1 - 2(0.083))(1 - 2(0.832)) = 0.223$

$r_{j,i}(-1) = 1 - r_{j,i}(+1)$

Now compute $q_{i,j}$'s from $r_{j,i}$'s :

$$\tilde{q}_{0,0}(+1) = (1 - p_0) \prod_{j' \in C_{0,0}} r_{j',0}(+1)$$

$$= (0.69) r_{2,0}(+1) = (0.69)(0.223) = 0.154.$$

and

$$\tilde{q}_{0,0}(-1) = p_0 \prod_{j' \in C_{0,0}} r_{j',0}(-1)$$

$$= (0.31) r_{2,0}(-1) = (0.31)(0.777) = 0.241.$$

This means,

$$q_{0,0}(+1) = \frac{0.154}{0.454 + 0.241} = 0.39$$

and

$$q_{0,0}(-1) = \frac{0.241}{0.154 + 0.241} = 0.61$$

Finally, compute the APP's:

Note: $\tilde{Q}_i(+1) = \tilde{q}_{i,j}(+1) \cdot r_{j,i}(+1)$, which means

$$\tilde{Q}_0(+1) = \tilde{q}_{0,0}(+1) \cdot r_{0,0}(+1)$$

$$= (0.154) \cdot (0.32) = 0.0493$$

and

$$\tilde{Q}_0(-1) = \tilde{q}_{0,0}(-1) \cdot r_{0,0}(-1)$$

$$= (0.241) \cdot (0.68) = 0.164.$$

This yields the APP

$$Q_0(+1) = \frac{0.0493}{0.0493 + 0.164} = 0.23$$

and

$$Q_0(-1) = \frac{0.164}{0.0493 + 0.164} = 0.77$$

The other Q_i's can be computed similarly.

Probabilistic decoding.

- The most significant feature of this decoding scheme is that the computation per digit per iteration is independent of block length.
- Average number of iterations required to decode is bounded by a quantity proportional to the log of the log of the block length.