Summary: In this lecture, we discuss Gallager's original hard decision decoding schemes as well as some variations of them. We also discuss some encoding schemes of LDPC codes having lower complexity as compared to systematic encoding. Finally, we discuss Gallager's construction and the Mackay-Neal method for design of LDPC codes

References:

Chapter 2, *Iterative Error Correction*, Sarah Johnson

Chapter 5, *Channel Codes: Classical and Modern*, William E. Ryan and Shu Lin

Introduction. In Gallager's 1963 paper introducing LDPC codes, he also presented two message-passing based algorithms for hard-decision decoding of LDPC codes. These have been named Gallager's Decoding Algorithm A and B. Both these decoding schemes perform reasonably in practice. We will detail these algorithms as well a few variations of the latter. We will invoke the standard notation for regular LDPC codes - (ω_c, ω_r), denoting the column and row weights respectively of the sparse parity check matrix.

Gallager's Decoding Algorithm A: We discuss Gallager's Decoding Algorithm A over the BSC channel. It is a hard-decision, message passing based decoding algorithm. Consider a (ω_c, ω_r) LDPC code and let the received codeword be denoted y. Then the algorithm proceeds as below:

1. Denote y_i as the i^{th} received bit and N_i as the neighbourhood of node i - the set of all check/bit nodes connected to bit/check node i. Transmit y_i to N_i for the i^{th} bit node.

$$\mu_{i\to k}^0 = y_i \qquad \forall k \in N_i$$

2. In intermediate iterations, at any check node j,

$$\mu_{j\to i} = \bigoplus_{k\in y/i} \mu_{k\to j} \qquad \forall i \in N_j$$

   Here $\oplus$ denotes summation modulo 2.

3. In intermediate iterations, at the bit node i, $\forall j \in N_i$,

$$\mu_{i\to j} = \begin{cases} y_i + 1 & \text{if } \forall k \in N_i/j, \quad \mu_{k\to i} = y_i + 1 \\ y_i & \text{otherwise} \end{cases}$$

   This means that for a particular check node j, if all other check nodes (connected to bit node i) unanimously show disagreement between their transmitted bit estimates $\hat{y}_i$ and received value y_i, the message is transmitted as $\hat{y}_i$.

4. At every iteration, compute the estimate for the codeword, $\hat{y}$:

$$\hat{y}_i = \begin{cases} y_i + 1 & \text{if majority of incoming messages from check-nodes are } y_i + 1 \\ y_i & \text{otherwise} \end{cases}$$

   Ties are broken based on the channel bit-flip probability, p.

5. If at any iteration, the estimated codeword satisfies the parity condition, i.e. if $\mathbb{H}\hat{y}^t = 0$, terminate, and return $\hat{y}$ as the estimate of the transmitted codeword.
   If not, repeat steps (2) to (5) until $\mathbb{H}\hat{y}^t = 0$ or upto a maximum number of iterations.

From the above analysis, since we do not invoke the channel transition probabilities (except in case of a tie), we can identify that the decoder is sub-optimal.

Gallager's Decoding Algorithm B: Gallager's Decoding Algorithm B for the BSC follows the same lines as Algorithm A. However, the message sent from the bit node to the check node changes. We define a sequence t_l indexed by the iteration number that gives the threshold for the number of check nodes connected to a bit node with contradicting message values to the value of the bit node. The bit node message computation in the B Algorithm is illustrated below: At bit node i, $\forall j \in N_i$

$$\mu_{i\to j} = \begin{cases} y_i + 1 & \text{if at least } t_l \text{ of check-nodes } \in N_i \text{ send } y_i + 1 \\ y_i & \text{otherwise} \end{cases}$$

l is the iteration number. In general, Gallager's Decoding Algorithm B performs better than Gallager's Decoding Algorithm A. Below, we discuss a variation of Gallager's Algorithm B.

Variation of Gallager's Decoding Algorithm B Let y be the received message. Denote the LDPC matrix associated with the code by $\mathbb{H}$. The variation can be computed in steps as sequentially illustrated below:

1. Compute the syndrome, $s = \mathbb{H}y^t$.
2. For each bit i, find f_i the number of failed checks it contributes in.This can be interpreted as an elementary measure of the probability of incorrectness of a bit.
3. At every iteration, simply flip the bit that has the maximum associated f_i.
4. If $\mathbb{H}y^t = 0$, terminate. If not, repeat steps (2) to (4)

Overall, the decoding complexity reduces from exponential to linear complexity because, on fixing the degrees of bit and check nodes, we need to perform a constant number of operations (that does not scale with n) per iteration.

Encoding Encoding of LDPC codes can simply be performed by multiplying the message vector with the generator matrix:

$$c = \underset{1\times k}{m} \; . \; \underset{k\times n}{\mathbb{G}}$$

However, the encoding complexity of such an approach is $\mathcal{O}(kn)$. If $k \propto n$, the complexity is $\mathcal{O}(n^2)$. With the decoding complexity being almost linear, quadratic complexity for encoding would limit the scaling of code length in practical applications. Consider the sparse parity check matrix associated with the LDPC code:

$$\mathbb{H} = \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{sparse}$$

Denote the parity check matrix corresponding to systematic encoding as $\mathbb{H}_{std}$. $\mathbb{H}_{std}$ is dense with high probability and hence the generator matrix in standard form is dense with high probability:

$$\mathbb{H}_{std} = \underbrace{\begin{bmatrix} P_{n-k\times k} & I_{n-k} \end{bmatrix}}_{dense}$$

$$\mathbb{G}_{std} = \begin{bmatrix} I_k & P^t_{k\times n-k} \end{bmatrix}$$

Thus, systematic encoding cannot be performed in linear time complexity. The scope for more efficient encoding schemes leveraging the sparse structure of LDPC matrices is discussed below:

We first consider the sparse form of the parity check matrix. We permute it's rows and separately permute the columns corresponding to the message and parity bits. This corresponds to reordering the bit and check nodes and does not modify the underlying code.

$$\mathbb{H} = \left[\begin{array}{c|c} \overleftrightarrow{\text{shuffle columns}} & \overleftrightarrow{\text{shuffle columns}} \end{array}\right] \updownarrow \text{shuffle rows}$$

$$\underbrace{\qquad\qquad}_{\text{message bits}} \quad \underbrace{\qquad\qquad}_{\text{parity bits}}$$

Rows and columns are shuffled as mentioned above, to get the form:

$$\mathbb{H} = \left[\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right] \begin{array}{l} \} \, n-k-g \text{ rows} \\ \} \, g \text{ rows} \end{array}$$

Here, A and C corresponds to message bits, while B, T, D and E correspond to parity bits. By construction, T is a lower triangular matrix.
Multiply A, B, T by ET^{-1} and subtract from the lower g rows to give

$$\mathbb{H} = \left[\begin{array}{c|c|c} A & B & T \\ \hline C - ET^{-1}A & D - ET^{-1}B & 0 \end{array}\right]$$

$$\mathbb{H} = \left[\begin{array}{c|c|c} A & B & T \\ \hline C' & D' & 0 \end{array}\right]$$

D' is guaranteed to be invertible as even after all these transformations, there are $n-k$ independent parity constraints in total of which a subset are covered by D' This guarantees that it is full rank.
Let us consider codewords with a virtual partition in the parity bits:

$$c = \begin{bmatrix} m & p_1 & p_2 \end{bmatrix}$$

By the parity constraint, we have:

$$\mathbb{H}.c^t = \left[\begin{array}{c|c|c} A & B & T \\ \hline C' & D' & 0 \end{array}\right] \begin{bmatrix} m & p_1 & p_2 \end{bmatrix}^t = 0$$

Which gives us:

$$Am^t + Bp_1^t + Tp_2^t = 0 \tag{1}$$

and

$$C'm^t + D'p_1^t + 0 = 0 \tag{2}$$

Solving (2) gives:

$$p_1^t = (D')^{-1}C'm^t$$

Solving (2) is of $\mathcal{O}(g^2)$ complexity. Plugging this back into (1), we can solve for p_2 with linear ($\mathcal{O}(n)$) complexity by forward substitution (since T is lower triangular). The overall efficiency of encoding depends on g. If g can be made small, the overall encoding complexity becomes nearly linear.

Gallager's Construction of LDPC codes for regular LDPC codes Using the standard notation for LDPC codes - (ω_c, ω_r), denoting the column weight and row weight respectively of the parity check matrix, we begin construction of the parity check matrix by defining $\mathbb{A}_0$ ($\frac{n}{\omega_r} \times n$) as below:

$$A_0 = \begin{bmatrix} \underbrace{1 \quad \cdots \quad 1}_{\omega_r} & \underbrace{1 \quad \cdots \quad 1}_{\omega_r} & \cdots \end{bmatrix}_{\frac{n}{\omega_r} \times n}$$

Define A_i as an arbitrary column permutation of A_0. A_i would once again have column weight 1 and row weight ω_r.

$$A_i = \pi_i(A_0)$$

We can design a complete parity check matrix from A_0 and its permutations as:

$$\mathbb{H} = \begin{bmatrix} \pi_0(A_0) \\ \pi_1(A_0) \\ \vdots \\ \pi_{\omega_c - 1}(A_0) \end{bmatrix}_{\omega_c . \frac{n}{\omega_r} \times n}$$

Since each of the $\pi_i(A_0)$ has column weight of 1, vertical concatenation would make the resultant $\mathbb{H}$ matrix have column weight ω_c. Such a construction performs reasonably well for regular LDPC codes. However, we are not guaranteed that the generated $\mathbb{H}$ matrix is full rank.

$$\omega_c . n = (n - k) . \omega_r$$

$$\implies \frac{k}{n} = 1 - \frac{\omega_c}{\omega_r}$$

Since in general, the constructed parity check matrix could have redundant parity checks, we can guarantee that a code designed with rate R would as such have:

$$R \geq 1 - \frac{\omega_c}{\omega_r}$$

Thus, we define the **design rate** of the code as:

$$\delta = 1 - \frac{\omega_c}{\omega_r}$$

with the condition that the rate of the LDPC code is at least equal to the design rate of the code (given a design rate, we can define a family of LDPC codes, all with constant ratio of column and row weights). We will next discuss another method for designing LDPC codes, the Mackay-Neal method.

Mackay-Neal Method for construction of LDPC codes: The Mackay-Neal method for constructing parity check matrices is a method for designing sparse parity check matrices. It can be understood as a column-wise sequential construction of the low density parity check matrix. At every iteration, a column of the $\mathbb{H}$ matrix is constructed:

1. At the i^{th} iteration, observe those locations in the current column under consideration that *do not* correspond to rows (check nodes) that already have ω_r associated bit nodes (i.e. ω_r 1's in that row).
2. We randomly choose ω_c locations from these positions (i.e. arbitrarily choose ω_c of these check nodes) and assign 1's in these locations.
3. Repeat this process, until all the columns have been constructed.

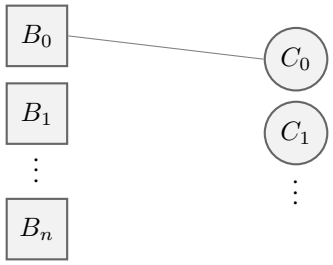
$$\mathbb{H} = \left[\begin{array}{ccccc|c|} \cancel{0} & \cancel{1} & \cancel{1} & & \cancel{0} & \cancel{0} & \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \\ 1 & 1 & 0 & \cdots & 0 & 0 & \\ 0 & 1 & 1 & \cdots & 0 & 1 & \end{array}\right] \longrightarrow \text{Row already has } \omega_r \text{ 1 's}$$

filled columns

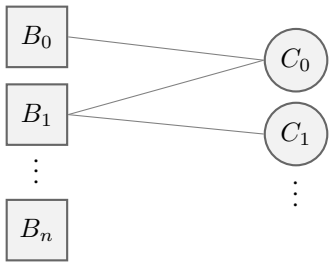
new column

In terms of a graph depiction of this process, this amounts to at the k^{th} iteration, considering the k^{th} bit node and adding ω_c edges to check nodes that do not already have ω_r edges to the other $k-1$ bit nodes.

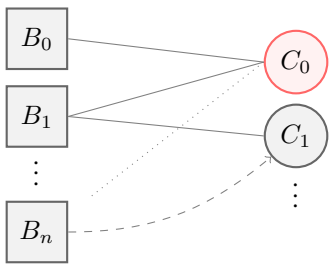
Begin with bit node 0, B_0 and choose ω_c check nodes to connect edges to:



For bit node 1, B_1, pick ω_c check nodes that are not saturated and connect edges to them:



For B_n, C_0 already has ω_r bit nodes connected to it, so it is not a viable check node to add an edge to. The dashed line indicates that an edge could be joined to C_1:



Other variations There are several other more effective algorithms for designing LDPC codes. A few among them are:

1. Progressive Edge filling algorithm
2. Bit-filling algorithm

In practice, both give very good codes.

The analysis of convergence and performance of decoding algorithms is carried out by studying the density evolution of the decoding algorithm. This will be the topic of discussion in the coming lectures.