

## LDPC codes for Enhanced Reliability in 5g NR networks



AY 2020-24

# GITAM UNIVERSITY

PROJECT ID – C3

A University should be a place of light, of liberty, and of learning.

## Department of Electrical Electronics and Communication Engineering




Project Team:

090 Dwarakanath Reddy  
400 Siva Charan  
457 Hemanth Kumar

Project Mentor:  
DR. Ambar Bajpai Sir

[www.gitamedu.com](http://www.gitamedu.com)



Photo	Track	Roll No	Name
	EECE	0090	Dwarakanath Reddy
	EECE	0457	Hemanth Kumar
	EECE	0400	Siva Charan

## 3





# **CONTENTS**

---

**INTRODUCTION**

---

**ABOUT LDPC**

---

**OBJECTIVE**

---

**FLOW CHARTS**

---

**ENCODING & DECODING**

---

**ADVANTAGES&DISADVANTAGES**

---

**COMPARISON**

---

**CONCLUSION**

---



## INTRODUCTION:

Low-Density Parity-Check (LDPC) codes are vital in 5G communication systems, playing a key role in ensuring that data is transmitted reliably, even in the presence of noise and interference. As 5G networks push for higher data rates, lower latency, and better connectivity, the need for robust error correction becomes critical. LDPC codes excel in this area by significantly reducing errors, allowing the network to maintain high-quality data transmission. They are particularly effective in 5G because they can approach Shannon's limit—the theoretical maximum data rate with minimal errors—making them extremely efficient for modern communication needs. Their ability to handle high data volumes while maintaining integrity makes LDPC codes a preferred choice for 5G and future network architectures.

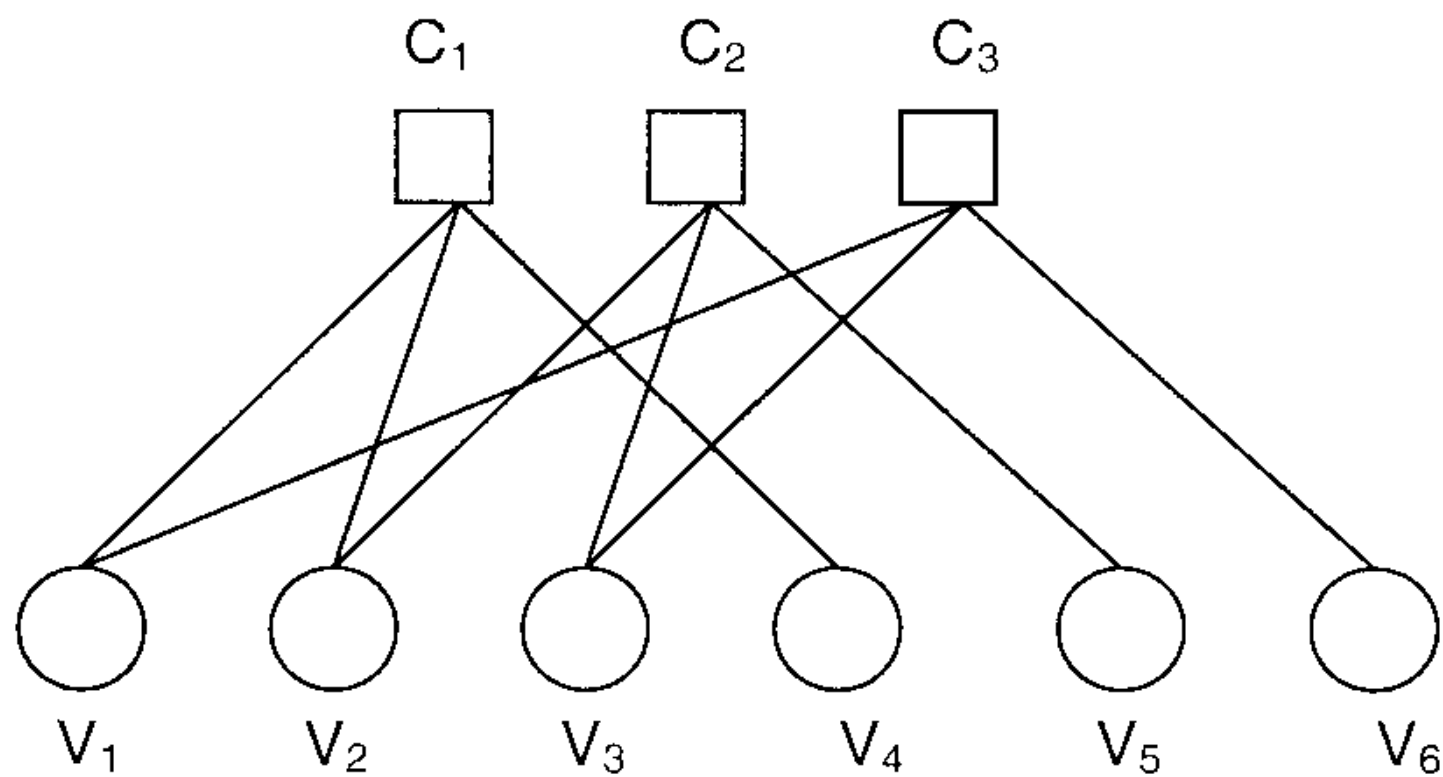


## ABOUT LDPC

- **Low Density parity check matrix**
- **Low density:** The number of “1s” in a matrix used for encoding is small compared to the number of “0s”. This matrix helps in error correction and having fewer “1s”.
- **Parity check:** Parity refers to checking whether the number of “1s” in a set of bits is odd or even. It helps in detecting and correcting errors in data.
- **Definition:** LDPC (Low-Density Parity-Check) is a type of error-correcting code used to detect and fix mistakes in data transmission. It works by adding extra bits to the original data, which helps in checking if any errors occurred during transmission. The "low-density" part means that the connections between these extra bits and the original data are sparse, making the code efficient and fast to decode.



- **Tanner graph:** A Tanner graph is a special type of graph used to represent error-correcting codes, like LDPC codes. It is made up of two types of nodes: variable nodes (representing data bits) and check nodes (representing parity checks). The connections (edges) between these nodes show how the data bits are checked for errors.
- **Sparse matrix:** A sparse matrix is a matrix (a grid of numbers) where most of the elements are zero. Only a small number of the entries have non-zero values.
- **Generative matrix:** A generator matrix is a special matrix used in coding theory to create codewords (which are the data plus extra bits for error correction) from the original data bits. It defines how to take the original data and turn it into a longer string of bits that can be sent or stored with error-correction capabilities.





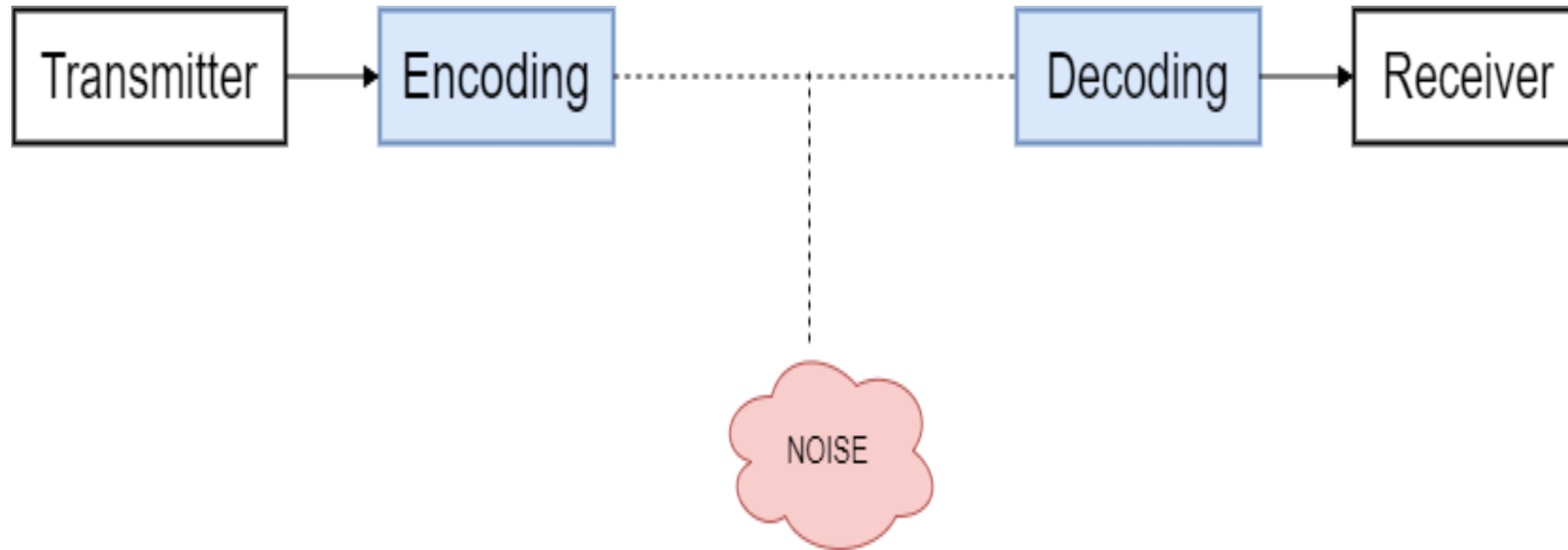


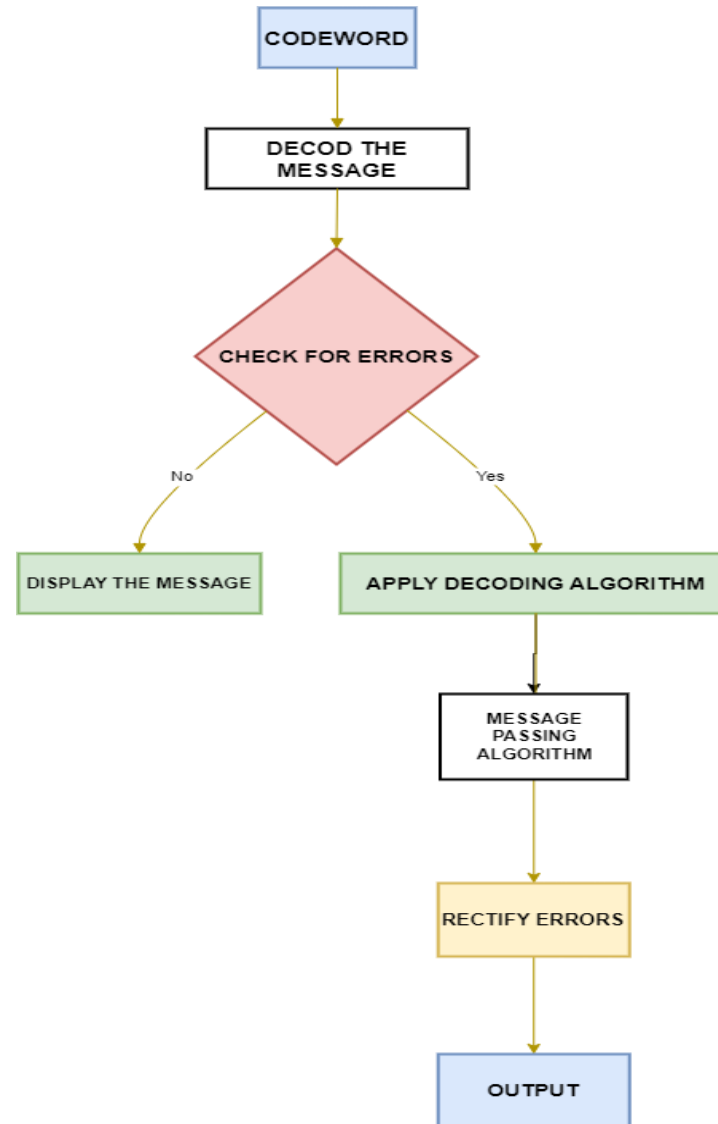
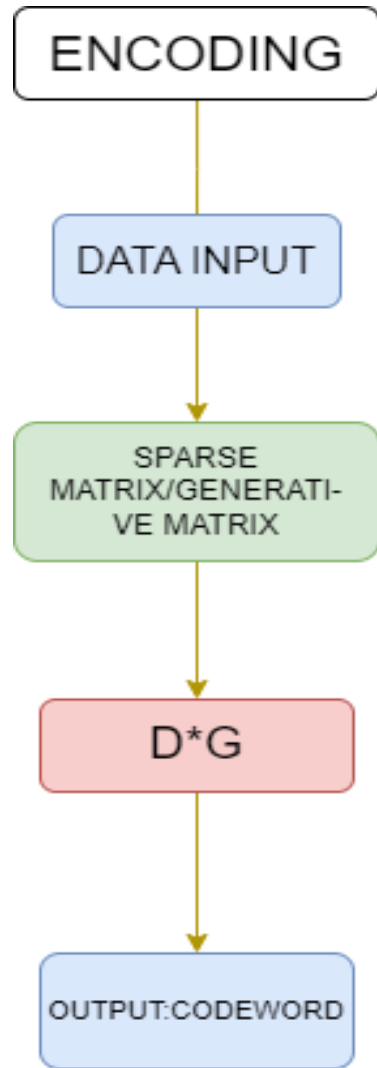
## **OBJECTIVE:**

- **Implement LDPC Encoding and Decoding:**  
Write code to implement the LDPC encoding process, where you add redundancy to the data, and the decoding process, where you use this redundancy to correct errors in the received data.



## BLOCK DIAGRAM







# ENCODING THE DATA

- **Definition:** Converting the message into binary format.
- **Main goal:** Creating parity bits and combining them with the original message using matrices.
- **Encoding in LDPC can be done in two ways:**
  - Generator matrix
  - Sparse matrix
- **Steps to perform:** (using Generative matrix)
  - Step 1:** Start with the original message in binary format.
  - Step 2:** Apply the generator matrix to the message.
  - Step 3:** Perform matrix multiplication to generate the encoded message with paribits.

## CODE SNIPPETS:

### How to encode the message using generative matrix:

```
% Define the data (6-bit)
data = [1 0 1 0 1 0];

% Define the generator matrix G (6x9)
G = [1 0 0 0 0 0 1 0 1;
     0 1 0 0 0 0 0 1 0;
     0 0 1 0 0 0 1 1 0;
     0 0 0 1 0 0 0 0 1;
     0 0 0 0 1 0 1 0 0;
     0 0 0 0 0 1 0 1 1];

% Encode the data (data * G)
encoded_data = mod(data * G, 2);

% Display the encoded data (codeword)
disp('Encoded data (codeword):');
disp(encoded_data);
```

- How **Sparse matrix** and **Generative matrix** are formed:

```
e/GEN.m
% Define the parity-check matrix H (must be full-rank)
H = [1 1 0 0 1 0;
     0 1 1 0 0 1;
     1 0 1 1 0 0];

% Partition H as [P | I] where P is a 3x3 matrix and I is a 3x3 identity matrix
P = H(:, 1:3); % Parity part
I = eye(3);    % Identity matrix

% Derive G matrix: G = [I | P']
G = [I P'];

% Display the generated generator matrix G
disp('Generated Generator Matrix G:');
disp(G);
```

```
% Set dimensions for the parity-check matrix H (m x n)
m = 4; % number of parity bits (rows)
n = 10; % number of total bits (columns)

% Generate a random sparse binary matrix
% Set sparsity by limiting the number of ones
H = zeros(m, n);
for i = 1:m
    % Randomly select positions for '1's in each row
    ones_pos = randperm(n, 4); % e.g., 4 ones per row for simplicity
    H(i, ones_pos) = 1;
end

% Display the generated parity-check matrix H
disp('Generated Parity-Check Matrix H:');
disp(H);
```



# DECODING THE DATA

- **Definition:** Retrieving the original message from received message and parity bits.
- There are Different types of decoding algorithms:
  - Message passing algorithm**
  - Sum product algorithm**
  - List decoding.....**



## MESSAGE PASSING ALGORITHM:

Belief propagation (or message passing algorithm) is a method used to solve certain types of problems, especially in graphical models like Bayesian networks or factor graphs. It's particularly useful when we want to figure out the probabilities of different events based on the relationships between variables.

### STEPS:

- **Graph Representation:** The system is represented as a graph where nodes are variables, and edges show how these variables are related.
- **Messages:** Each node passes "messages" to its neighboring nodes. These messages contain information about what one node "believes" about its neighboring node's state or probability.
- **Update Beliefs:** As messages are exchanged, each node updates its own belief about its possible states based on the messages it receives from its neighbors.
- **Iterative Process:** This exchange of messages continues iteratively until the beliefs (or probabilities) at each node stabilize.
- **Final Beliefs:** Once the process finishes, the nodes will have a pretty good estimate of the true probabilities for each variable.



## • CODING PART:

```
% Original encoded message (for reference)
encoded_message = [1 0 1 0 1 0 1 1 1]; % Example encoded message

% Received message with errors
received_message = [1 0 1 0 0 1 1 0 0]; % Message with errors

% Adjusted Parity-check matrix (H) for a 9-bit message
H = [1 1 0 0 1 0 1 0 0;
     0 1 1 0 0 1 0 1 0;
     1 0 1 1 0 0 0 0 1];
```

```
% Calculate the initial syndrome
syndrome = mod(H * received_message', 2);

% Maximum iterations for the message passing algorithm
max_iter = 10;

% Decoding process (iterative)
decoded_message = received_message; % Start with received message
```

```
for iter = 1:max_iter
    if all(syndrome == 0) % If no errors detected, stop
        break;
    end

    % Identify the bit to flip based on the syndrome
    for i = 1:size(H, 2)
        % Calculate the contribution of each bit to the syndrome
        error_contribution = mod(H(:, i)' * syndrome, 2);

        if error_contribution == 1
            % Flip the bit in the received message
            decoded_message(i) = 1 - decoded_message(i);

            % Recalculate the syndrome after flipping the bit
            syndrome = mod(H * decoded_message', 2);

            % Break after correcting one bit and recalculate the syndrome
            break;
        end
    end
end
```

```
% Display corrected original message
corrected_original_message = decoded_message(1:6);
disp('Corrected Original Message:');
disp(corrected_original_message);
```



## Advantages

- **High Error Correction Capability:**  
LDPC codes can correct a significant number of errors, making them effective for reliable data transmission over noisy channels.
- **Near Shannon Limit:**  
They perform very close to the theoretical maximum efficiency of data transmission, which means they use bandwidth effectively.
- **Efficient Decoding:**  
LDPC codes can be decoded using efficient algorithms, allowing for faster processing in practical applications.

## Disadvantages

- **Complexity in Encoding:**  
The encoding process is more complex than some other coding methods, requiring more computational resources.
- **Longer Latency:**  
Decoding can take time, especially for longer codes, which may introduce delays in data transmission.
- **Memory Usage:**  
LDPC codes may require more memory for the parity-check matrix, which can be a limitation in resource-constrained environments.

# COMPARISON OF LDPC CODES WITH TURBO CODES

## LDPC CODES

- Excellent **error correction**, approaches Shannon limit, especially for large block lengths
- Generally more efficient, supports **parallel decoding**, faster for large code lengths
- Easier to implement in hardware, flexible in **code design** (length and rate)

## TURBO CODES

- Strong error correction, performs well near Shannon limit, particularly for **shorter block lengths**
- More complex, **iterative decoding** leads to longer processing times in some cases
- More challenging to implement due to recursive nature, but flexible in terms of code rates



# Conclusion

LDPC (Low-Density Parity-Check) codes are a key component in enhancing the reliability and efficiency of 5G communication systems. They offer superior error correction capabilities, which are crucial for maintaining data integrity in high-speed networks. LDPC codes help achieve lower latency and higher throughput, essential for supporting advanced 5G applications like IoT, autonomous vehicles, and ultra-reliable low-latency communication. By optimizing these codes, we can ensure robust and efficient data transmission, making them an integral part of the future of wireless communication.



# *Thank You!*

**Presented By,**  
**Dwarakanath Reddy [090]**  
**Hemanth Kumar [457]**  
**Siva Charan[400]**