
University of Padova

Channel Coding

2017 - 2018

# LDPC using WiMAX standards

*Student name:*
Kalakonda Srikanth Reddy

*Student Number:*
1178232

August 19, 2018

# Contents

# 1 Abstract

Channel coding enables the effective transmission of data over a noisy channel.Low density parity check codes (LDPC) are an old class of efficient channel codes that have been ignored until recently. When LDPC codes were originally introduced in the 1960's, they exceeded the hardware capabilities at the time, and were considered a theoretical novelty. However, over the past ten years, there has been a renewed interest in LDPC codes due to technological advancements that finally allowed efficient implementations to be realized. LDPC codes have become part of several state-of-the-art wireless standards, including the IEEE 802.16e standard that is behind mobile-WiMAX.

In this problem report, an encoder for encoding and decoder for decoding the LDPC codes for 802.16e is developed. Using this , simulations of this code were conducted and used to produce bit error rate plots.

# 2 Introduction

Communications presents a simple problem. Some information needs to be sent from one entity to another through some channel. This information that is received must match the original information. The challenge is to find how to recover the original information if some sort of noise or errors occurs during the transmission process. There are simple ways of ensuring data is correctly heard such as repeating the transmission. The problem with this is it takes twice as long to convey the information than if the information was heard without errors. Error control coding seeks to maximize the throughput while ensuring the information is transmitted correctly.

In order for communication systems to function, the receiver must obtain the data the transmitter had sent. To achieve this, some type of redundancy must be added to combat errors that occur across a channel. The errors can be caused by interference from other signals as well as noise in the channel itself. The challenge is that it is difficult to obtain an acceptable error rate while not compromising the data rate. The careful balance between error rate and data transmission rate is the focus of most research in coding theory. The goal is to achieve maximal data rates while maintaining inconsequential error rates. There are a variety of different methods for obtaining these goals.

## 2.1 WiMAX Technology

WiMAX (Worldwide Interoperability for Microwave Access) is *a standards-based technology enabling the delivery of last mile wireless broadband access as an alternative to cable and DSL.*WiMAX is based upon std. IEEE 802.16e-2005 which provides multiple PHY and MAC options. The standard specifies different FEC options:

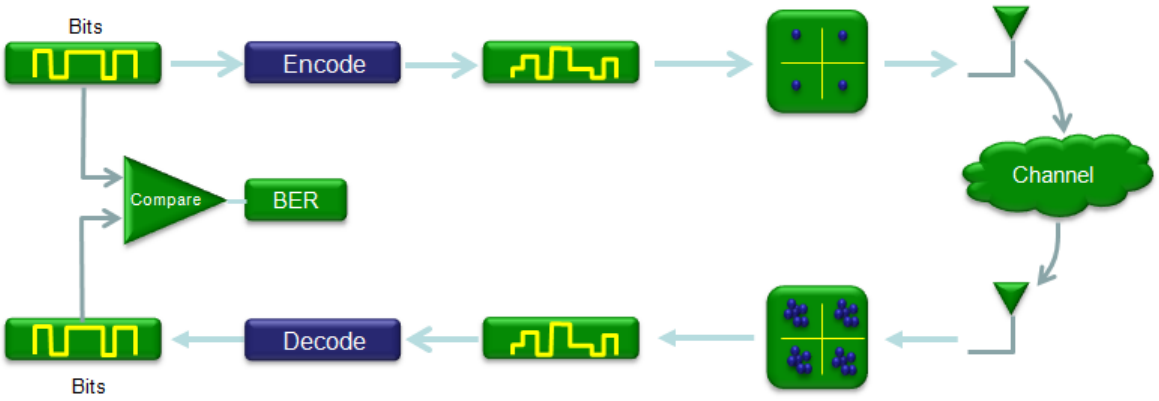
- Convolutional Coding (CC)
- Convolutional Turbo Coding (CTC)
- Block Turbo Coding
- **Low Density Parity Check codes - LDPC**

## 2.2 LDPC codes

Low Density Parity-check (LDPC) codes were first proposed in the early 1960's [Ga63]. These codes are called low density because of the sparse nature of the paritycheckmatrix. The parity-check matrix contains a small number of ones which lends to the name. These early codes were widely neglected because they were too complex to implement using the hardware that was available at the time. Therefore until decades later, research did not focus on LDPC codes. Recently, LDPC codes have proven to efficiently transmit data. The code has a great deal of flexibility since the code can support a range of rates and accuracy based on the needs of the system.

The code uses a sparse matrix which allows only storing the non-zero entries and locations instead of storing all the entries in the matrix. Different ways of implementing the LDPC codes allows them to outperform the turbo code in some cases. Most notably, LDPC codes were chosen over turbo codes in the DVB-S2 standard for satellite coding. LDPC codes have been shown to come very close to the Shannon capacity. The outstanding performance that can achieve rates close to the Shannon channel capacity, as well as its greater acceptance, are motivation for further study of LDPC codes.

### 2.3 Channel coding procedure - Block diagram




# 3 Channel coding procedure - Implementation

The standard specifies a LDPC code for each combination of rate $R$ and codeword length $n$. There are:

- 6 possible rates (1/2, 2/3A, 2/3B, 3/4A, 3/4B, 5/6)
- 19 possible codeword lengths

We implemented in Matlab the encoder/decoder couple for some codes. In particular we considered:

- Rate (1/2 )
- 2 different codeword lengths (576, 1344 bits)

## 3.1 LDPC codes

LDPC codes are linear block codes, and they are often in systematic form. The fundamental codes can by used for different code rates and packet sizes. The $H$ matrix defines each set of LDPC codes. The $H$ matrix is of size $m \times n$. where n is the length of the code and $m = n - k$. are the number of parity bits.

Each code in the set of LDPC codes is specified by a parity check matrix $H$ of size $m \times n$.

$H$ is defined as:

$$H = \begin{bmatrix} P_{0,0} & P_{0,1} & \dots & P_{0,n_b-1} \\ P_{1,0} & P_{1,1} & \dots & P_{1,n_b-1} \\ \vdots & \vdots & & \vdots \\ P_{m_b-1,0} & P_{m_b-1,1} & \dots & P_{m_b-1,n_b-1} \end{bmatrix}$$

where $P_{i,j}$ is a $z \times z$ permutation matrices or a $z \times z$ zero matrix with $z = \frac{n}{n_b} = \frac{m}{m_b}$ and $n_b = 24$.

The permutations used are circular right shifts. $H$ is expanded from a base model matrix $H_{bm}$ of size $m_b \times n_b$:

- each positive entry $p(i,j)$ is replaced by a $P_{i,j}$ matrix with circular shift size equal to $p(i,j)$
- each blank or negative entry p(i,j) is replaced by a $P_{i,j}$ zero matrix

The standard defines $H_{bm}$ for the largest codeword length ($n = 2304$) of each code rate. $H_{bm}$ for a different codeword length $n'$ is obtained by scaling the shift sizes proportionally

$$p'(i,j) = \begin{cases} p(i,j) & p(i,j) \leq 0 \\ \left\lfloor \frac{p(i,j) z_f}{z_0} \right\rfloor & p(i,j) > 0 \end{cases}$$

where $z_0 = \frac{2304}{n_b}$ and $z_f = \frac{n'}{n_b}$.

## 3.2 $H_{bm}$ Matrix

The following is the $H_{bm}$ matrix corresponding to the code rate we used.

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 94 | 73 | -1 | -1 | -1 | -1 | -1 | 55 | 83 | -1 | -1 | 7 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 27 | -1 | -1 | -1 | 22 | 79 | 9 | -1 | -1 | -1 | 12 | -1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 24 | 22 | 81 | -1 | 33 | -1 | -1 | -1 | 0 | -1 | -1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 61 | -1 | 47 | -1 | -1 | -1 | -1 | -1 | 65 | 25 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 39 | -1 | -1 | -1 | 84 | -1 | -1 | 41 | 72 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 46 | 40 | -1 | 82 | -1 | -1 | -1 | 79 | 0 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 | -1 | -1 |
| -1 | -1 | 95 | 53 | -1 | -1 | -1 | -1 | -1 | 14 | 18 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 | -1 |
| -1 | 11 | 73 | -1 | -1 | -1 | 2 | -1 | -1 | 47 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 |
| 12 | -1 | -1 | -1 | 83 | 24 | -1 | 43 | -1 | -1 | -1 | 51 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 |
| -1 | -1 | -1 | -1 | -1 | 94 | -1 | 59 | -1 | -1 | 70 | 72 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| -1 | -1 | 7 | 65 | -1 | -1 | -1 | -1 | 39 | 49 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 43 | -1 | -1 | -1 | -1 | 66 | -1 | 41 | -1 | -1 | -1 | 26 | 7 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 |

## 3.3 Matrix Generation

Given the chosen values for the rate $R$ and codeword length $n$, the script `matrix.m`:

- generates the matrix $H$ using the corresponding base matrix $H_{bm}$
- computes and saves the matrices $M_1$, $M_2$ and $M_3$

At the begin of each simulation, the matrices $M_1$, $M_2$ and $M_3$ are loaded and used during the encoding procedure.

## 3.4 Encoding procedure

Encoding allows for the successful transmission of information over a noisy channel. In practice, most channels are viewed as AWGN. This is a simplified view of a typical channel since interference usually plays a role too, but AWGN channels usually just treat the interference as noise. The key to producing a successful signal is to introduce redundancy to the information which is being sent. Redundancy gives the receiver a way of recovering the original signal even if some of the bits are altered because of the noisy channel. The type of redundancy is what gives different encoding procedures different results in terms of rate and SNR.

For an efficient encoding, H is divided into the form:

$$H = \begin{bmatrix} A_{(m-z)\times Rn} & B_{(m-z)\times z} & T_{(m-z)\times(m-z)} \\ C_{z\times z} & D_{z\times k} & E_{z\times(m-z)} \end{bmatrix}$$

We define: $M_1 = ET^{-1}A + C$, $M_2 = T^{-1}A$, $M_3 = T^{-1}B$.

Given the infoword $u$, we compute:

- $p_1^T = M_1 u$
- $p_2^T = M_2 u + M_3 p_1^T$

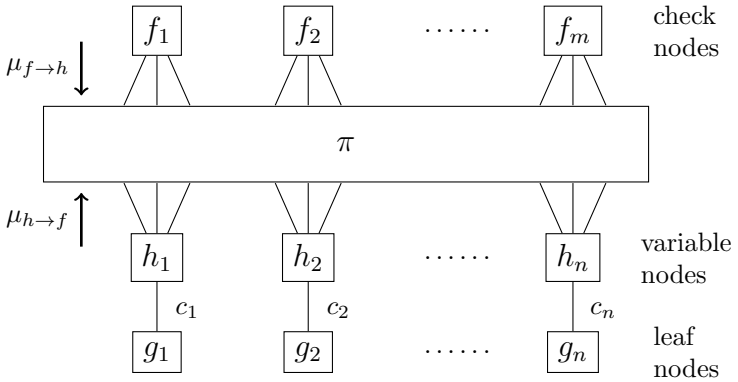
The corresponding codeword is $v = [u, p_1, p_2]$

## 3.5 Decoding binary LDPC

Decoding is essentially the opposite of encoding. Instead of starting with the information and adding redundancy, the decoder takes the received signal, which contains

information, redundancy information and/or errors, and tries to find the most likely message. The challenge for the decoder is to discriminate between the correct information and the errors created by the channel. The decoder takes the received signal and tries to match it to a valid codeword. The decoder can use different means for matching the received signal to the codeword. It may depend upon the probability of a codeword being transmitted, the distance to the closest codeword, or use a lookup table to determine the error and codeword which needs to be recovered. Section 4 will show the implementation decoding.

The decoding is performed using the Message Passing Algorithm (MPA) in the logarithmic domain. The MPA is based on this factor graph:




## 3.6 Message Passing Algorithm

**Initialization:** Messages $\mu_{h\to f}$ are initialized with the channel a-priori knowledge $g$.

**Schedule:**

1. run message passing on check nodes
2. run message passing on variable nodes
3. perform ongoing marginalization:

$$\hat{c}_l = \begin{cases} 0 & , \mu_{h_l\to c_l} + \mu_{g_l\to c_l} \geq 0 \\ 1 & , otherwise \end{cases}$$

4. go back to step 1 until the maximum number of iterations is reached or a codeword is found

## 3.7 Leaf nodes

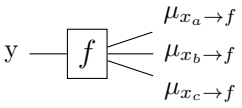
Under the hypothesis of equally probable input symbols, the messages from leaf to variable nodes are:

$$g_l = -\frac{2r_l}{\sigma_w^2}$$

where $r_l$ is the l-th received symbol and $\sigma_w^2$ is the noise variance.

## 3.8 Check nodes

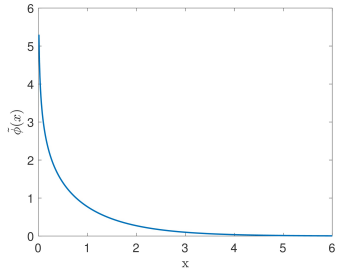
The update rule for the check nodes is:




$$\mu_{f \to y} = \tilde{\phi}\left( \sum_i \tilde{\phi}(|\mu_{x_i \to f}|) \right) \prod_i sign(\mu_{x_i \to f})$$

where:

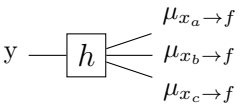
$$\tilde{\phi} = -\log\left( tanh\left( \frac{1}{2}x \right) \right) = \tilde{\phi}^{-1}$$




## 3.9 Variable nodes

The update rule for the variable nodes is:

$$\mu_{h \to y} = \sum_i \mu_{x_i \to h}$$




# 4 Decoder implementation

The decoder is implemented by the function `decode` which outputs the estimate $\widehat{\mathbf{u}}$ of the transmitted infoword $\mathbf{u}$, given the received vector $\mathbf{r}$ and the noise variance

$\sigma_w^2$.

- the vector `g` contains the messages exiting the leaf nodes and going upwards
- the matrix `mu_hf` contains the messages variable → check
- the matrix `mu_fh` contains the messages check → variable
- the vector `mu_hg` contains the messages exiting the leaf nodes and going downwards

`decode` implements the Message Passing schedule described above:

**Initialization:** compute `g` and initialize `mu_hf`

**while** max n of iterations is not reached AND codeword is not found

- update `mu_fh`

$$\tilde{\phi}\ (\text{—}\texttt{mu\_hf}\text{—}) = \begin{bmatrix} \tilde{\phi}(|\mu_{h_1\to f_1}|) & \tilde{\phi}(|\mu_{h_1\to f_2}|) & \dots & \tilde{\phi}(|\mu_{h_1\to f_m}|) \\ \vdots & \vdots & & \vdots \\ \tilde{\phi}(|\mu_{h_n\to f_1}|) & \tilde{\phi}(|\mu_{h_n\to f_2}|) & \dots & \tilde{\phi}(|\mu_{h_n\to f_m}|) \end{bmatrix}$$

$$\texttt{tmp3} = \left( \begin{bmatrix} \sum_1^n \tilde{\phi}(|\mu_{h_i\to f_1}|) \\ \vdots \\ \sum_1^n \tilde{\phi}(|\mu_{h_i\to f_m}|) \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \right) .* H - \tilde{\phi}(|\texttt{mu\_hf}|)^T$$

$$sign\ (\texttt{mu\_hf}) = \begin{bmatrix} sign(\mu_{h_1\to f_1}) & sign(\mu_{h_1\to f_2}) & \dots & sign(\mu_{h_1\to f_m}) \\ \vdots & \vdots & & \vdots \\ sign(\mu_{h_n\to f_1}) & sign(\mu_{h_n\to f_2}) & \dots & sign(\mu_{h_n\to f_m}) \end{bmatrix}$$

$$\texttt{mu\_fh} = \tilde{\phi}(\texttt{tmp3}) .* \left( \begin{bmatrix} \prod_1^n sign(\mu_{h_i\to f_1}) \\ \vdots \\ \prod_1^n sign(\mu_{h_i\to f_m}) \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \right) .* sign(\texttt{mu\_hf})^T$$

- Update `mu_hf` and `mu_hg`

$$\texttt{mu\_hf} = \left[ \left( \begin{bmatrix} \sum_1^m \mu_{f_i\to h_1} \\ \vdots \\ \sum_1^m \mu_{f_i\to h_n} \end{bmatrix} + \texttt{g} \right) \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \right] .* H^T - \texttt{mu\_fh}^T$$

$$\texttt{mu\_hg} = \begin{bmatrix} \sum_1^m \mu_{f_i\to h_1} & \sum_1^m \mu_{f_i\to h_2} & \dots & \sum_1^m \mu_{f_i\to h_n} \end{bmatrix}$$

- marginalize

$$\hat{c} = (\texttt{mu\_hg} + \texttt{g}) < 0$$

The $\tilde{\phi}$ function is truncated to avoid over/underflow errors:

$$\tilde{\phi}(x) = \begin{cases} 12.5 & x < 10^{-5} \\ -\log\left(tanh\left(\frac{1}{2}x\right)\right) & 10^{-5} \leq x < 50 \\ 0 & x \geq 50 \end{cases}$$

# 5 Simulation

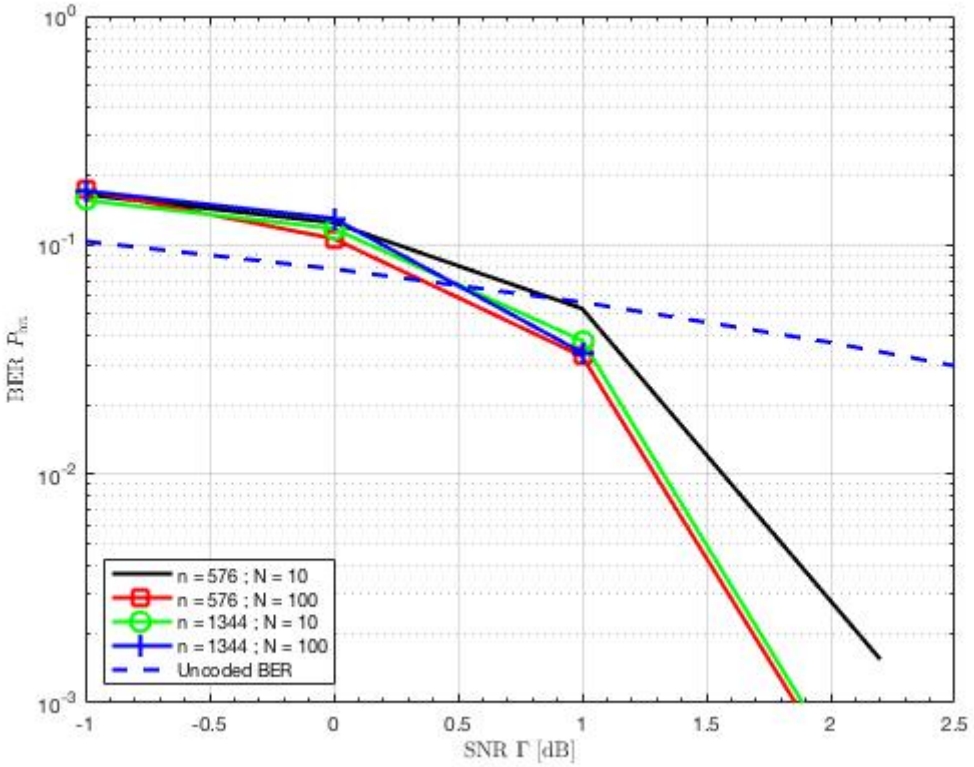
The script `simulate.m` runs a simulation campaign to test the performance of a code in terms of $P_{bit}$ vs $SNR$.

For each $SNR$ value to test, the simulation follows this schedule:

1. generate a random infoword $\mathbf{u}$
2. encode $\mathbf{u}$ to obtain $\mathbf{c}$
3. modulate $\mathbf{c}$ to obtain $\mathbf{c_{mod}}$
4. add AWGN noise with variance $\sigma_w^2$ to obtain $\mathbf{r}$
5. decode $\mathbf{r}$ to obtain $\widehat{\mathbf{u}}$
6. compute the number of errors comparing $\mathbf{u}$ and $\widehat{\mathbf{u}}$
7. go back to step 1 until the total number of errors or the number of transmitted packets reach the threshold

# 6 Results

Performance of binary LDPC ($n = 576$, $R = 1/2$) and ($n = 1344$, $R = 1/2$) for different number of iterations.This can be checked by running script `results.m`.


$10^0$
$10^{-1}$
$10^{-2}$
$10^{-3}$
BER $P_{\text{en}}$
-1
-0.5
0
0.5
1
1.5
2
2.5
SNR $\Gamma$ [dB]
n = 576 ; N = 10
n = 576 ; N = 100
n = 1344 ; N = 10
n = 1344 ; N = 100
Uncoded BER

———o x o———