IEEE 802.11 LDPC Codes: Implementation and Performance Analysis

Alessandro Lucatello[†]

***Abstract*—Noadays it is trivial to say that the wireless communication is used everywhere (e.g., telecommunication, IoT, health care, etc.). One of the problems with wireless communication is interference, which causes packet errors. Error-correcting codes (ECC) can assist with this reality-related problem. This project is focused on providing to the reader an implementation of the LDPC codes with different design choices in the decoding part. This report explores the theory behind the LDPC codes and provides a summary of the decision made regarding the code. It is possible to investigate the behavior of two different channel models: AWGN and Markov, with the combined effect of interleaving. It also compares the results of various tests to determine the most convenient design option based on the selection system.**

***Index Terms*—LDPC, Error Correction, IEEE 802.11, Message Passing, Interleaving.**

I. Introduction

The noise that is injected into the channels will negatively impact wireless communication networks. Channel codes are the essential part of wireless communication systems which help in detection and correction of errors due to the noise introduced in the channel. LDPC codes are close to the Shannon limit, which determines which codes are feasible, making them optimal codes. This means that for a given SNR value, they achieve a high spectral efficiency within the Shannon capacity limit. DVB-S2/DVB-T2/DVB-C2 (Digital video broadcasting, second generation), DMB-T/H (Digital video broadcasting), Wimax (IEEE 802.16e standard for microwave communications), 802.11-2020 (Wi-Fi standard) are the few standards in which the LDPC codes are used [1].

The project's main goal is to implement and simulate this error-correcting code, as well as to investigate its decoding complexity and bit error rate for different combinations of decoding cycles and channel model.

This project includes a simulator that incorporates the signal generation, encoding, channel, and decoding pipeline. This simulator was built using the *C++* programming language, and the code can be found in the following GitHub repository.

The report follows the following list of Sections: in the Sec. II is described a general scheme of the simulator. Sec. III discusses the project's theory to provide an overview of how the code is implemented. Sections IV and V describe the test setup and results, respectively. Finally, Section VI demonstrates how the Markov model and interleaving work.

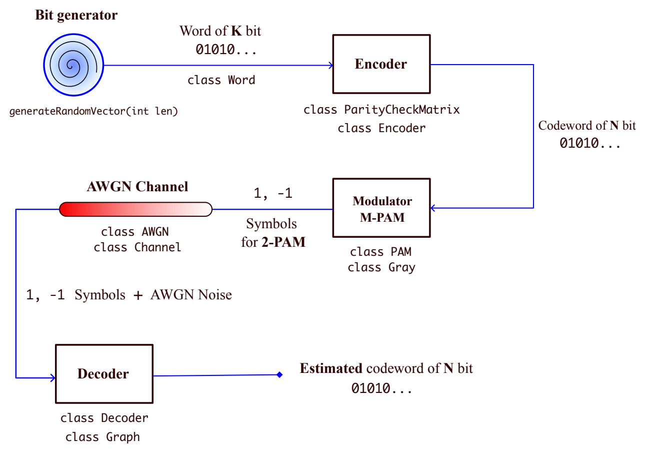


Fig. 1: Diagram that shows the different blocks of the simulator

II. Simulator In A Nutshell

As mentioned in Sect. I, the project includes a simulator that follows the IEEE 802.11 standard [2] for the encoding and decoding processes. Section 19.3.11.7 of the IEEE 802.11 standard describes the encoding process, while Annex F, located at the end of the document, defines the parity-check matricies.

The code creation guideline is to assign each class in *C++* to a specific assignment. Fig. 1 depicts the entire pipeline, with the corresponding classes for each block. Every class is listed here, along with their respected assignments:

- `Word`: It represents the word of bits. This class includes some useful methods for reading and creating a word.
- `ParityCheckMatrix`: Creates the parity-check matrix according to the rules outlined in the IEEE 802.11 standard, which will be explained in the following sections.
- `Encoder`: Produces the generating matrix G using Gaussian elimination. This matrix is responsible for creating the encoded word.
- `PAM` & `Gray`: They work together to assign the appropriate modulation symbol to a group of bits.
- `AWGN` & `Channel`: Represent the channel by sampling a zero-mean Gaussian random variable and adding the sampled value to the modulated symbol.

[†]Department of Information Engineering, University of Padova, email: alessandro.lucatello@studenti.unipd.it

- `Graph`: Generates a graph between equality nodes and check nodes using the parity-check matrix. It also incorporates the Message Passing algorithm.
- `Decoder`: Implement soft-decoding at the receiver using two versions of bit interleaved coded modulation (BICM).

All the methods of these classes are explained, with some comments, in the "`.h`" files of the project.

III. In The Depths Of Theory

This Section provides a detailed theoretical explanation of the different blocks that were introduced in Sec. II with some nice diagrams. Another important goal of this section is to try to link the theory to the most challenging part of the code, which is critical to understanding the project (the Sub-Sections titled "Understanding the code" are created to achieve this goal). This type of `font` indicates that all the words are code-related functions, variables or files.

The purpose of the different blocks of the diagram is explained previously in Sect. II. Now we will delve into the theory behind the simulator.

A. LDPC Encoder

The LDPC encoder is systematic, so it encodes an information block $\mathbf{u} = (i_0, i_1, ..., i_{k-1})$ by adding $n-k$ parity bits. The codeword $\mathbf{c} = (i_0, i_1, ..., i_{k-1}, p_0, ..., p_{n-k-1})$ is represented as a vector of `int`, with each element being 0 or 1. $\mathbf{H}$ is an $(n-k) \times n$ parity-check matrix and $\mathbf{H} \times \mathbf{c}^\top = 0$. To calculate the codeword $\mathbf{c}$, these steps must be followed:

1) **Creation of H parity-check matrix**: According to [2], a parity check matrix is a partitioned matrix composed of $Z \times Z$ submatrices. These submatrices are either cyclic-permutations of the identity matrix or null submatrices. Annex F of [2] defines $\mathbf{H}$ as follows: a table of real numbers that indicates a cyclic-permutation matrix $\mathbf{P}_i$ is obtained from the $Z \times Z$ identity matrix by cyclically shifting the columns to the right by i elements. The "–" entries denote null (zero) submatrices. For example, see Fig. 2.

Remark III.1. It is important to reason about the parameters shown in Fig. 2:

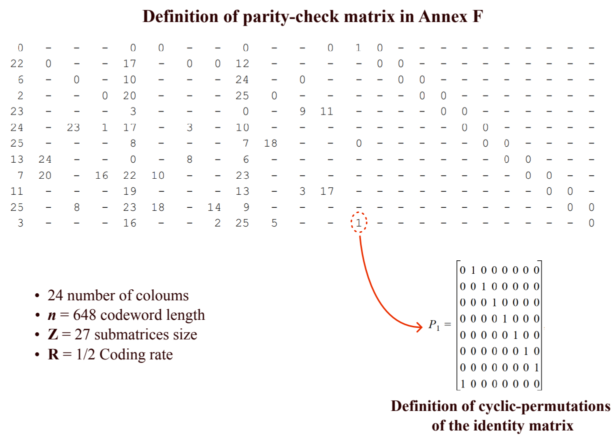
- $n = 24 \cdot Z = 648$ is the number of the columns of $\mathbf{H}$.
- $R = \frac{k}{n}$. So, $k = 324$ and $n - k = 324$ is the number of rows of $\mathbf{H}$.

It is possible to tune the R and Z according to n to obtain 12 different parity-check matrices, in accordance with the IEEE 802.11 standard [2].

2) **Generating matrix G creation**: The $\mathbf{H}$ matrix must first be subjected to Gaussian elimination in order to produce the following form:

$$\mathbf{H}' = \begin{bmatrix} A & I_{N-K} \end{bmatrix} \tag{1}$$

where I_{N-K} is the $(n-k) \times (n-k)$ identity matrix. $\mathbf{G}$ can now be obtained by taking a $n \times k$ matrix and inserting


Definition of parity-check matrix in Annex F
24 number of coloums
n = 648 codeword length
Z = 27 submatrices size
R = 1/2 Coding rate
Definition of cyclic-permutations of the identity matrix

0	–	–	–	0	0	–	–	0	–	–	0	1	0	–	–	–	–	–	–	–	–	–	–
22	0	–	–	17	–	0	0	12	–	–	–	–	0	0	–	–	–	–	–	–	–	–	–
6	–	0	–	10	–	–	–	24	–	0	–	–	–	0	0	–	–	–	–	–	–	–	–
2	–	–	0	20	–	–	–	25	0	–	–	–	–	–	0	0	–	–	–	–	–	–	–
23	–	–	–	3	–	–	–	0	–	9	11	–	–	–	–	0	0	–	–	–	–	–	–
24	–	23	1	17	–	3	–	10	–	–	–	–	–	–	–	–	0	0	–	–	–	–	–
25	–	–	–	8	–	–	–	7	18	–	–	0	–	–	–	–	–	0	0	–	–	–	–
13	24	–	–	0	–	8	–	6	–	–	–	–	–	–	–	–	–	–	0	0	–	–	–
7	20	–	16	22	10	–	–	23	–	–	–	–	–	–	–	–	–	–	–	0	0	–	–
11	–	–	–	19	–	–	–	13	–	3	17	–	–	–	–	–	–	–	–	–	0	0	–
25	–	8	–	23	18	–	14	9	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
3	–	–	–	16	–	–	2	25	5	–	–	1	–	–	–	–	–	–	–	–	–	–	0

$$P_1 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \end{bmatrix}$$

Fig. 2: Example of parity-check matrix

in the first k rows an identity matrix and inserting the first $n-k$ columns of $\mathbf{H}'$ in the last $n-k$ rows.

$$\mathbf{G} = \begin{bmatrix} I_K \\ A \end{bmatrix} \tag{2}$$

Remark III.2. The $\mathbf{H}'$ matrix is less sparse than the $\mathbf{H}$ matrix, but since the $\mathbf{H}'$ matrix is only used during encoding, this has no bearing on the decoding procedure.

3) **Creation of the codewords**: The encoding is a multiplication between the generating matrix $\mathbf{G}$ and the information block $\mathbf{u}$ (a vector of k bits) as follows:

$$\mathbf{c} = \mathbf{G} \cdot \mathbf{u} = \begin{bmatrix} \mathbf{u} \\ \mathbf{r} \end{bmatrix} \tag{3}$$

The redundancy bits, or $\mathbf{r}$, are what allow error detection and correction. During the decoding process, the systematic code that was previously mentioned is very helpful. The information bits can actually be extracted from the first k elements of the encoded word $\mathbf{c}$.

B. PAM Modulation

In order to enable effective transmission across a variety of media, modulation maps some bits into a symbol that corresponds to a particular combination of amplitude, phase, or frequency of a carrier wave. The waveform set consists of:

$$s_n(t) = \alpha_n h_{Tx}(t), \quad \alpha_n = 2n - 1 - M \tag{4}$$

where $n = 1, 2, .., M$. The equations above show that PAM waveforms are created by modulating the amplitude of the pulse shape h_{Tx}. α_n can assume this set of values:

$$\begin{aligned} \mathcal{A} &= \{\alpha_1, \ldots, \alpha_M\} = \\ &= \{-(M-1), -(M-3), \ldots, (M-3), (M-1)\} \end{aligned} \tag{5}$$

Choosing $h_{Tx} = sinc(t)$, the E_h is equal to 1. This is a convenient choice, but it is difficult to implement with practical circuits [3]. This decision is suitable for this project since it has no effect on the deconding complexity. In the constellation, this vector (point of constellation) represents the waveform:

$$s_n = \left[\alpha_n \sqrt{E_h}\right] \tag{6}$$

So, this is the constellation:

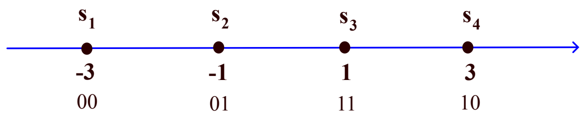


Fig. 3: Example of constellation for a 4-PAM

The Gray bit mapping is shown in Fig. 3 below the constellation vectors. Gray's words differ from the adjacent words by one bit.
In the example of the 4-PAM modulation, the bits per symbol are $b = log_2 4 = 2\ bits$, and $M = 4$ represents the modulation order and the number of symbols.
If I take the example of the Sub-Sect. III-A, the encoded word has $n = 648\ bits$. The following steps should be taken to obtain the modulated word using a 4-PAM modulation. Encoded words are divided into groups of two bits. To obtain the constellation symbols, apply Gray's map to each group of two bits.
The resulting modulated word, composed of constellation symbols, is injected into the channel.

C. AWGN Channel

In the transmission process, each modulated symbol is subjected to additive white Gaussian noise (AWGN) when passing through the channel. For every symbol transmitted, a distinct and independent noise sample is generated. This noise, modeled as a Gaussian random variable, is added directly to the transmitted symbol. Each noise sample is real-valued and has a variance of $N_0/2$, where N_0 represents the noise power spectral density. This simulates the AWGN channel, ensuring that the received signal is a combination of the original transmitted symbol and the noise. Since each transmitted symbol is affected by noise independently of other symbols, the independent noise samples guarantee that the channel remains memoryless, closely simulating real-world communication scenarios.

D. Message Passing

This subsection outlines the decoding method applied to a received block modulated with 2-PAM. Using this modulation order, each symbol carried only one bit. In the following subsection, this process is extended to incorporate Binary Interleaved Coded Modulation (BICM). By doing this, the decoding benefits from the increased spectral efficiency offered by higher-order modulation schemes while retaining its effective binary structure. Using larger modulation orders, this method achieves better bandwidth utilization while still keeping the decoding process simple.

The network architecture that the Message Passing for the binary decoding will use is shown in Fig. 4. The first things that draw the eye are squares (nodes) and lines (links). There are three different types of nodes: check nodes, equality nodes, and symbol nodes. These can be seen arranged from top to bottom.

Each link has two "streams": one that runs from top to bottom and the other in the opposite direction, as shown in Fig. 4 from the arrows. In the links, "flows" the $LLRs$. We use this value instead of propagating two messages for the bits 0 and 1 to reduce the computational complexity.

Understanding the code: Adjacency lists are used in C++ to realize the graph. Take as an example `AdjListCheckNodes`. This is a vector of vectors of pair vectors, and they represent the message realesed from the check nodes to the equality nodes.
The check nodes are indicated by the first index of the initial vector, while the position of the link connected to the selected check node is indicated by the second index of the initial vector's element, which is itself a vector. The second index's vector returns a pair of values. The first value is the selected link's destination, indicating an equality node. The second value represents the selected link's LRR.
Following the previous reasoning, it is trivial to understand that the links between g nodes and the equality node require the use of a vector.

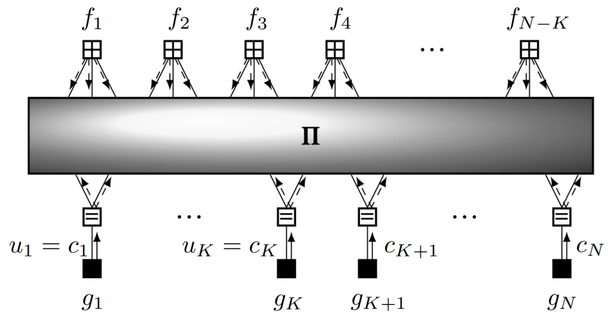


Fig. 4: Graphical representation of Message Passing [4]

Before beginning the iteration part of the Message Passing, the first thing to do is to calculate the LLR_{g_l} released from the symbol node, which has equality nodes as its destination (see Fig. 5 - **a**).
For calculate the LLR_{g_l} it is used the following equation:

$$LLR_{g_l} = \frac{2r_l}{\sigma_l^2} + \ln\left(\frac{p(u_l = 0)}{p(u_l = 1)}\right) \tag{7}$$

where, as in this instance, the ln term disappears for equally likely symbols. Instead, r_l is the symbol received after the AWGN channel at the receiver, and σ_l^2 represents the noise variance.

We use the following equation to update the LLRs released from the equality node, which has check nodes as its destination (see Fig. 5 - **b**):

$$LLR_y = LLR_a + LLR_b \tag{8}$$

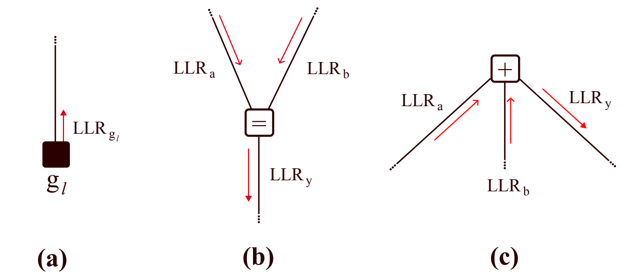


Fig. 5: LLR calculation for the different nodes

which is simply the sum of the incoming links in the equality nodes.

Remark III.3. Fig. 4 shows that for the first iteration of the decoding cycle, the two messages calculated with Eq. 8 are equal to LLR_{g_l}.

In Fig. 4, to update the three messages coming down from the check nodes to the equality nodes, use the following equation (see Fig. 5 - **c**):

$$LLR_y = \prod_i \text{sign}(LLR_i) \cdot \tilde{\phi}\left(\sum_i \tilde{\phi}(|LLR_i|)\right). \quad (9)$$

To calculate the LLR_y in Eq. 9, one of the three links connected to the check node is selected as the output link, while the remaining two links are the input links. The i index in Eq. 9 slides all incoming links in the check node.
After calculating LLR_y for the first output link, repeat the process for the remaining links.

To evaluate the $\tilde{\phi}$ function, use the following equation:

$$\tilde{\phi}(x) = \ln\left(\frac{e^x + 1}{e^x - 1}\right) \quad (10)$$

which represents the most computationally demanding part.

To obtain a possible codeword, perform a marginalization following this rule:

$$\hat{u}_l = \begin{cases} 0 & \text{if } LLR_g + LLR_= > 0 \\ 1 & \text{otherwise} \end{cases} \quad (11)$$

where $LLR_=$ is the sum of the $LLRs$ that come from the check nodes and enter in the equality node calculated with the Eq. 9. Instead, $\hat{u}_l$ is one of the bits that create the estimated codeword.

Save all of the possible marginalization results, and this is the estimated codeword. Looking for this codeword. If no codeword matches, continue the decoding process until a codeword is found.

Refer to [4] for more information on determining the various equations of the LLR.

E. BICM LDPC Decoder

In contrast to the binary LDPC Decoder, the modulation is M-PAM, with each symbol carrying more than one bit. In Fig. 6, an 8-PAM is used, with each symbol carrying 3 bits.

When the Binary LDPC Decoder and the BICM LDPC Decoder are compared, it is clear that a new network of links is formed between the equality nodes and the symbol nodes (g nodes), with one side connected to the equality nodes and the other to the new conform nodes (w nodes). The w nodes are connected to the symbol nodes via a single link.

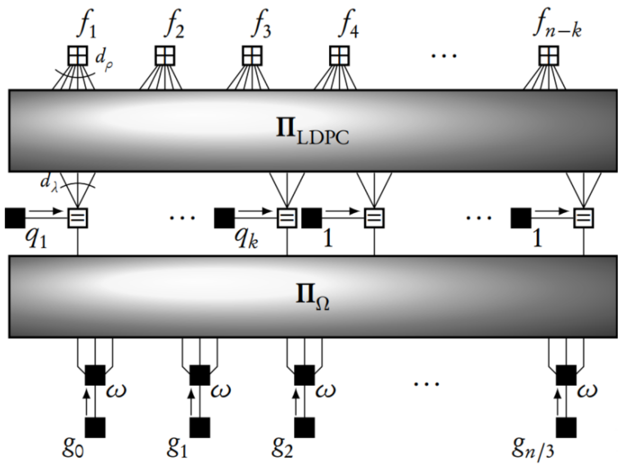


Fig. 6: Graphical representation for the BICM LDPC Decoder

This project use the network structure in Fig. 6 with three differences:

1) To keep things simple, in this project, each of the three upper links of the conform nodes w_i is connected to an equality node using this rule. The link $l = 0, 1, ..., log_2 M - 1$ of the nodes w_i with $i = 0, ..., \frac{n}{log_2 M}$ connects to the equality node with the index equal to $log_2 M \cdot i + l$.
This ensures that none of the conform nodes' links overlap.
2) The g_l nodes are not used. The r_l received symbols are linked directly to the conform nodes.
3) The presence of $q_1, ..., q_k$ nodes that are used for marginalization. They represent the a priori probability of K source symbols. In our case, all symbols have the same probability, so the LLR value of these nodes can be ignored during marginalization.

The decoding cycle begins with updates to the conform nodes' outgoing links. Fig. 6 requires updating the three upper links connected to the equality nodes.

Conform nodes use a different method to update the output links. As in Fig.7, select the first link (y) as the output link and the other as input links (x_a, x_b). Use Eq. 12, where the fixed bit in the function map is the bit of the output link and the other bits ($x_a, x_b, ..$) are the possible permutations of the

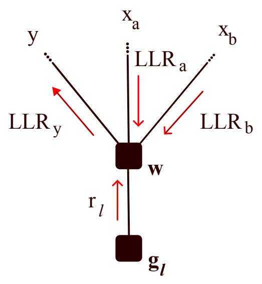


Fig. 7: Conform nodes representation

input bits.

$$LLR_{w\to y} = \\ = \ln\left(\frac{\sum_{x_a,x_b,...} g_l(\text{map}(0,x_a,x_b,...))e^{(1\oplus x_a)\text{LLR}_a+\cdots}}{\sum_{x_a,x_b,...} g_l(\text{map}(1,x_a,x_b,...))e^{(1\oplus x_a)\text{LLR}_a+\cdots}}\right) \tag{12}$$

Observing Eq. 12, the following functions can be noticed:

- The *map* function takes a sequence of bits as input and maps it to the corresponding constellation symbol using Gray's code.
- The g function has the following expression:

$$g_l(d_l) = \exp\left(-\frac{(r_l - d_l)^2}{(2\sigma_w)^2}\right) \tag{13}$$

Remark III.4. In the first iteration of the decoding cycle, the exponential term of Eq. 12 is equal to 1 because the $LLRs$ are initially set to 0.

Understanding the code: Here, the code for Eq. 12 is analyzed.
Beginning with the calculation of the inputs of the *map* function within the g function. The vector `possible_permutation` stores the possible permutations of 1 and 0 for $(x_a, x_b, ...)$ variables. Four nested cycles are used to select the position of the output link, the value of the output link, choose one of the possible permutations in the corresponding vector, and read the element of `possible_permutation` vector (a vector itself). Inside the last cycle, the value of the output link is temporarily saved in the vector `vector_map_temp`, and the values of the vector `possible_permutation` are saved in the remaining positions. If the output link has a value of 1, it is stored in the vector `vector_map_0`, otherwise in the vector `vector_map_1`.
The same procedure is followed for values found in the exponential of Eq. 12. The values of the `possible_permutation` vector are temporarily saved in the vector `exponent_map_temp` and then depending on the value of the output link, it is saved in the vector `exponent_map_0` or `exponent_map_1` (a vector of vectors).
The vector `output_link` saves the position of the output link for the corresponding element in the vectors `vector_map_0` and `vector_map_1`. The elements of this vector are useful as input of the function `calculateLLRwNodes`, which returns the sum of the LLRs for the links connected to the w nodes, excluding the output link.
With the previously calculated values, we can calculate the results for Eq. 12.

This project includes two different versions of the BICM LDPC decoder. One has a fast decoding cycle, while the other has a slow decoding cycle. The major distinctions are the following:

- **Fast Decoding Cycle**: After the first update of the conform nodes' outgoing links, this procedure repeats the steps used for Message Passing. The difference is that the marginalization occurs on the q nodes, while the value of the conform nodes' outgoing links remains unchanged. If the estimated codeword is invalid, repeat the binary decoding cycle on the upper network structure until convergence occurs.
- **Slow Decoding Cycle**: For the first iteration until marginalization, we repeat the Fast Decoding Cycle steps. The value of the messages that are passing from the equality nodes to the conform nodes is updated at the end of the first iteration, prior to marginalization.
  If the estimated codeword is not valid in the first iteration, the cycle must be repeated until convergence is achieved, beginning with the update of the conform nodes' outgoing links.

IV. Prior To Measurement

Before analyzing the results, this section describes the set-up and highlights some considerations for the various tests.

A. *BER performance of BICM Fast and Slow Decoding Cycle*

The maximum number of iterations for both cycles is fixed at 20. For what is stated in Sub-Sect. III-E for the first iteration of the cycle, the two cycles have the same path.
If the variance σ_w^2 is small (between 0.25 and 0.4), both cycles will converge (find a valid codeword) during the first or second iteration. If the value of the variance increases (greater than 0.8), the two cycles will never converge. A word that is close to a valid codeword can be obtained after 20 iterations, and it is nearly identical to the codeword obtained after 100 or 1000 iterations. So, 20 iterations are an acceptable choice for balancing a precise bit error rate with a reasonable decoding time.

A chunk of encoded words is considered when calculating the BER. The BER calculation consists of calculating the BER value for each encoded word and then taking the average. This allows for more accurate BER values.
For high variance values (between 1 and 3), changing the number of encoded words in a block does not have an effect on the BER value. So, the strategy is to have a large number of encoded words per block for low variance (high SNR) and a small number of encoded words per chunk for high variance.

B. *Decoding Complexity*

To measure complexity, the number of elementary operations performed to decode is used as a proxy, such as additions, multiplications, divisions, comparisons, table lookups, and so on, with each assigned an energy cost.
The exponential and logarithmic functions are the most costly. Also, the nested cycle is expensive.

V. Fast vs Slow Decoding Cycle

This section examines the results obtained for BER and decoding complexity for the two different decoding cycles.

A. *BER performance of BICM Fast and Slow Decoding Cycle*

Fig. 8 depicts a Cartesian coordinate system. The horizontal axis shows the amount of energy used for each bit of information in relation to the system's noise. It is a dimensionless quantity that is commonly expressed in decibels (dB).
As shown in Fig. 8, the bit error rate decreases with decreasing noise variance (so that E_b/N_0 increases). As can be seen, for high values of noise variance, the BER for the two decoding cycles are nearly identical. The differences appear when the noise variance is between 0.6 and 1, with the slow decoding cycle's BER decreasing before the fast decoding cycle's.
As a result, for values of E_b/N_0 ranging from 0.8 to 1.3, the BER of the slow decoding cycle is always greater than that of the fast decoding cycle.

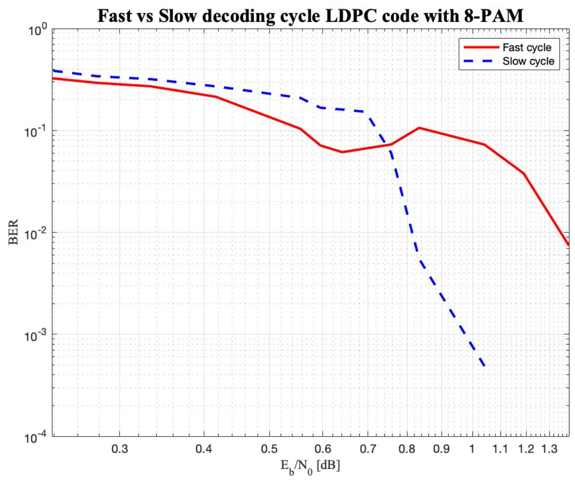


Fig. 8: Comparison of two-cycle BER as a function of SNR

This is an important finding that demonstrates that the BICM slow decoding cycle outperforms the fast decoding cycle for certain values of noise variance. This is what the theory predicts [4].
In Fig. 8, the value of 0 for the BER is not represented due to the logarithmic scale. If the simulation repeats the BICM for a large number of codewords (more than 100,000), it should find a BER value close to zero because it should find a codeword

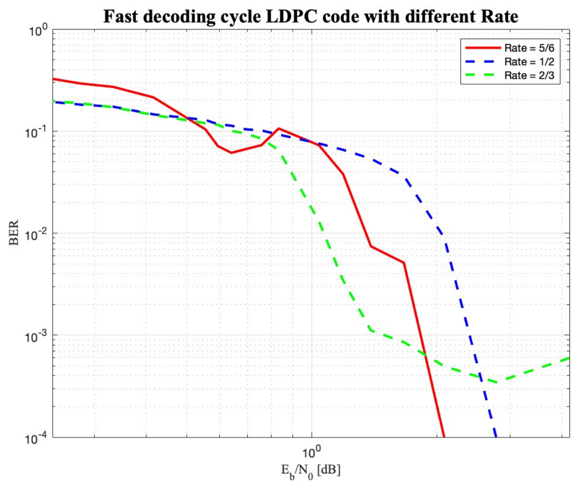


Fig. 9: Comparison of different BER as a function of SNR and rate variation

whose BER value is not zero.
Fig. 9 shows how the BER varying a function of SNR and coding rate R variation (defined in the IEEE 802.11 standard [2]).

B. *Decoding Complexity*

Fig. 10 illustrates a Cartesian coordinate system with the vertical axis representing the sum of operations multiplied by their costs.
It is clear that the slow cycle has higher costs than the fast cycle for almost all values on the horizontal axis. The costs remain the same for high values of E_b/N_0 (greater than 3.2 dB). The two cycles have similar values because they both end with a valid codeword in the first iteration of the cycle. For the fact that both cycles follow the same path during the first iteration, they have the same number of operations.

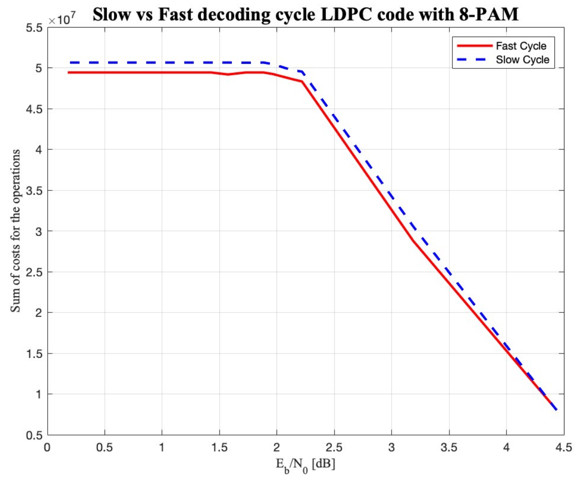


Fig. 10: Comparison of two-cycle decoding complexity

C. Final Showdown

Which one of the two cycles should I choose? Depends. For a better BER it is better to choose the slow cycle in all the situations. However, if complexity is a constraint, it is always preferable to use a fast algorithm that requires fewer operations for missing conform node updates after the first iteration. Remember that for low variance noise, both cycles have the same number of operations and BER performance.

VI. Markov Model and Interleaving

In this section, we introduce a new channel model, in addition to the AWGN channel, the Markov channel based on that. Furthermore, before sending the data, a technique known as interleaving is used to improve data transmission reliability, especially against burst errors.

This project includes the implementation of interleaving and the Markov channel model. The tests focus on how the BER behaves as the interleaving depth varies.

A. Markov Channel

It consists of 2-state Markov chain: the state 0 represents a good state and the state 1 represents a bad state. A good state has a low variance for noise, whereas a bad state has a high variance. Each states have its own Noise Power Spectral Density (PSD) (N_0^0 and N_0^1). The PSD for the bad state is higher than the PSD for the good state.
The transition probability is another defining feature of the Markov model. The probability of moving from one state to another (P_{tr}) is equal to 0.001, implying that the probability of remaining in the state is 0.999. A very low probability of moving to another state is chosen to lengthen the burst errors. The transition probability matrix, as shown below, describes the Markov model:

$$P = \begin{pmatrix} 1 - P_{tr} & P_{tr} \\ P_{tr} & 1 - P_{tr} \end{pmatrix} \tag{14}$$

It embeds what was said above.

B. Interleaving and Deinterleaving

The interleaving is done in the file `interleavingTest.cpp` (*interleaving_v1* red line in Fig. 11), which operates as follows:

1) Create a matrix with L (interleaving depth) rows and select the number of modulated words (columns) for each row. The end result is a matrix with modulated words as its elements.
2) Take the sub-matrix made up of all the rows and one column of the previous matrix. This sub-matrix is the interleaving matrix.
3) Read the columns of the interleaving matrix and inject them into the Markov channel model.
4) Deinterleaving involves taking the words from the channel and placing them column by column in a received matrix.
5) To read the deinterleaved words, look at the rows of the previously created matrix (received matrix). Each of these rows represents a modulated word that is sent at the start.

In the file `interleavingTest_v2.cpp` (*interleaving_v2* blue dashed line in Fig. 11), there is a different implementation for interleaving: instead of doing it in the sub-matrix, as previously explained, it is possible to do it in the matrix with modulated words as its elements. Both implementations use the same variance value when calculating the g function of the BICM's conform nodes (Eq. 13).
The third and final implementation of interleaving (*interleaving_v3* green dashed line in Fig. 11) follows the same procedure as *interleaving_v2*, but for the calculation of the g function of BICM's conform nodes (Eq. 13), different values of the variances are used, depending on the states in which the Markov model channel was.
Fig. 11 shows that the first implementation (*interleaving_v1*) performs best in terms of BER for interleaving depths less than 100. However, for values of interleaving depth greater than 100, the better implementation is the third (*interleaving_v3*) which brings the BER very close to zero.

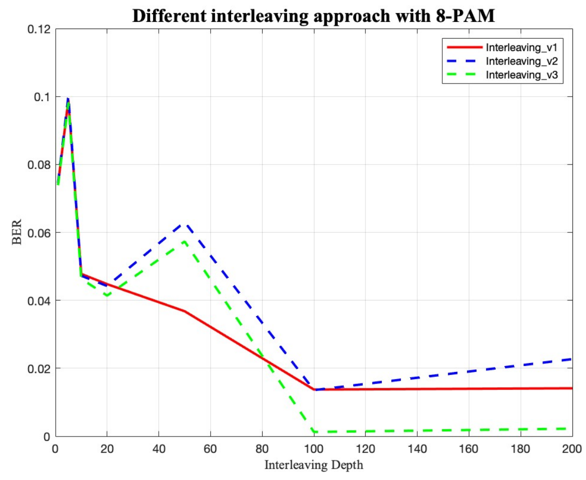


Fig. 11: A comparison of various interleaving approaches

VII. Concluding Remarks

This work implements and analyzes several network coding algorithms, with a specific focus on how these techniques improve data transmission efficiency in network systems. The primary responsibilities included developing, testing and validating the efficacy of various coding schemes, as well as understanding how they affect network throughput and robustness.

Network coding techniques improved data transmission by reducing redundancy and allowing networks to operate at near-capacity.

One significant limitation in this project is the assumption of ideal network conditions, which may not always hold true in practice. Future work could explore adaptive coding schemes that can dynamically adjust to changing network environments.

This project taught me how to design experiments to validate theoretical findings using practical simulations, which is a necessary skill for applying theoretical algorithms to real-world problems.

References

[1] E. a. Mrs. Channaveeramma. E, “Performance comparison of turbo coder and low-density parity check codes,” *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 2021.

[2] “Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,” *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[3] N. Benevenuto and M. Zorzi, *Principles of Communications Networks and Systems*, ch. 5, pp. 259–371. John Wiley Sons, Ltd, 2011.

[4] T. Erseghe, *Channel Coding*. Padova University Press, 2016.