Ukrainian Catholic University

Faculty of Applied Sciences

LA in coding theory

Linear Algebra final project report

Authors:
Katja Kovalchuk
Max Kutsenko
Roman Zaletskyy

06 March 2024

1 Introduction

When transferring any information, for example: user registration data on the service, processing of this data, transfer of this data to the server, the result of the transaction when paying on the service, data of the card from which the payment was made, - through the channel, errors may occur, which we have to correct so that the information is delivered correctly.

Also, encoding information and transmitting it through the channel for subsequent operations on it is very relevant for group and private chats, because the messages themselves must be transmitted without errors, since chat users must understand their content for further communication.

Another area where the correct transmission of information through the channel is very important is working with electronic money and building the security of banking systems in mobile applications, since each monetary transaction has its own hashed number, which, if deciphered, (correctly) can change the recipient and thereby cause great trouble to the bank. Therefore, in order to prevent such cases, it is important for us to code the information correctly and decipher it correctly, and if errors were detected, then correct them, if this is provided for by the algorithm itself, if not, then block access to this entity (transaction) in order to prevent hacking.

2 Problem

We want to try to solve the problem of decoding a message with errors that occur when sending a message from device to device. When the word decoding is mentioned, those who have studied discrete mathematics and probability theory (especially those who did the first task of the second lab) think of Hamming code system, which does a pretty good job of correcting errors, but it is not perfect. So let's take a closer look at it.

3 Hamming code system

3.1 Hamming code

Hamming code is a system that allows you to detect the number of errors when receiving coded information and when transmitting decoded information in the form of bits. Also, this system allows you to correct an error in one bit, by adding extra bits to the message, with the help of this system, you can encode the message by channel, and detect the number or, if possible, correct the errors in the received message.

3.2 Error detection and correction

Detecting and correcting channel errors is extremely important in today's data transmission world. The Hamming code system allows you to detect and correct one bit in a message by adding parity bits to a given message.

3.3 Mathematics approaches

3.3.1 Hamming code

1. We cut our message into lines of length 4 bits. (b1 b2 b3 b4)
2. We put parity bits in this message at positions 2^n, where n is a natural number (positions 1, 2, 4).
   (a) Parity of bit 1 is $P_1 = b1 \oplus b2 \oplus b3$.
   (b) Parity of bit 2 is $P_2 = b1 \oplus b3 \oplus b4$.
   (c) Parity of bit 3 is $P_3 = b2 \oplus b3 \oplus b4$.
3. Then we form our message (P1, P2, b1, P4, b2, b3, b4) and send it.

1. We receive our 7-length message.
2. Multiply this message on Hemming matrix (7, 4)

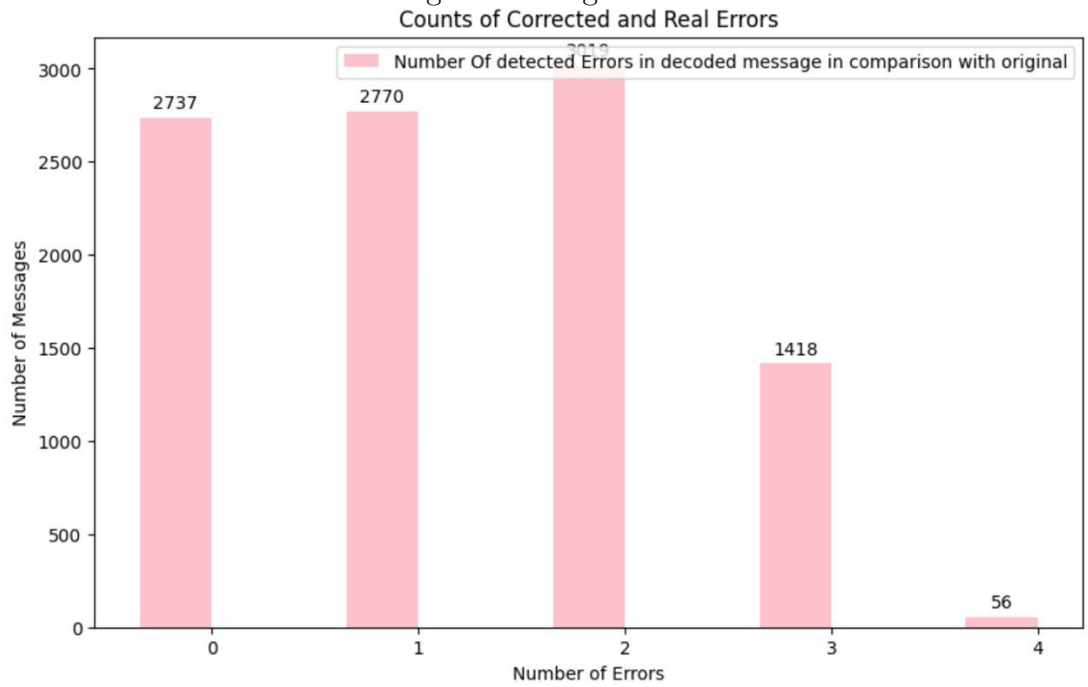
$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

   in form (bit of a message) * (column of a matrix) -¿

$$\text{P1} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + P2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + b1 \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + P4 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + b2 \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + b3 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + b4 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

3. Result of a multiplication will be a column of a matrix that represents a bit with error.
   (a) If a column is all zero - there is no error, if a combination of bits in a column is not in the matrix - there is more than 1 error.
4. Change the value in an error bit (if 1 - change to 0, if 0 - change to 1)
5. Remove purity bits from message
6. We get our message with corrected error.
7. In this plot, we can see the number of errors in the already decoded messages. This graph does not include errors in the parity bits. Its just check the number of bits

that coincide in decoded and original message.



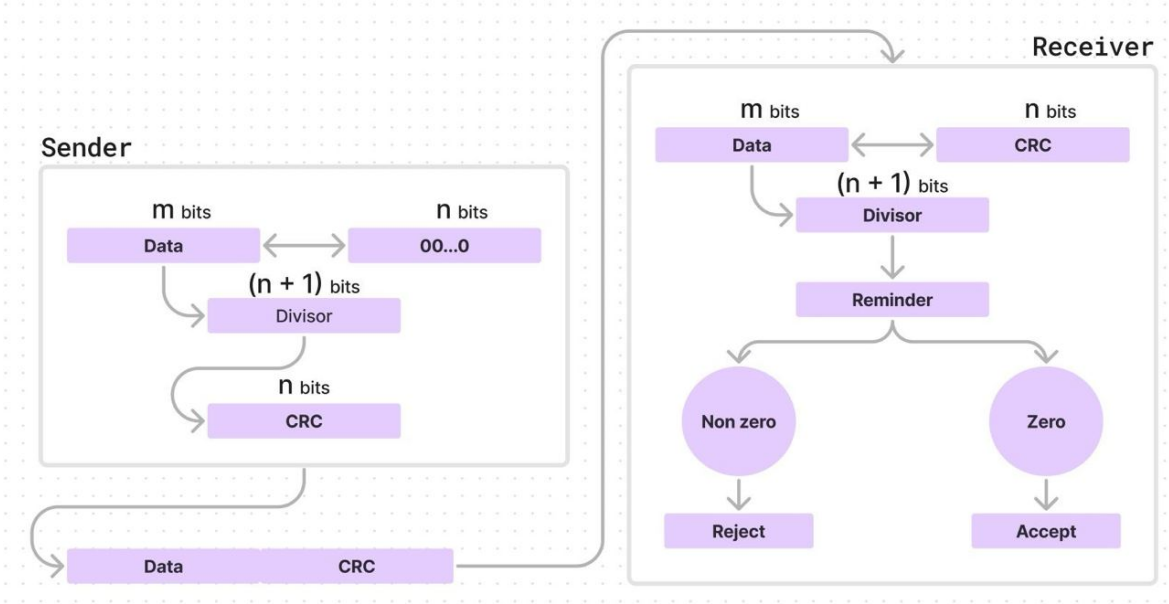
8. But there is one problem with Hamming code: it can correct only one error, or report that there are two errors, but not fix them. This is why this encoding is a bit bad, because if there are 3 errors, it may not even notice that there was an error and give the wrong result.

9. Why did we choose this algorithm as the main algorithm for the research? It is the easiest to implement and everyone knows it. Yes, it has its disadvantsges, such as correcting only one error in a message, but it's pretty good.

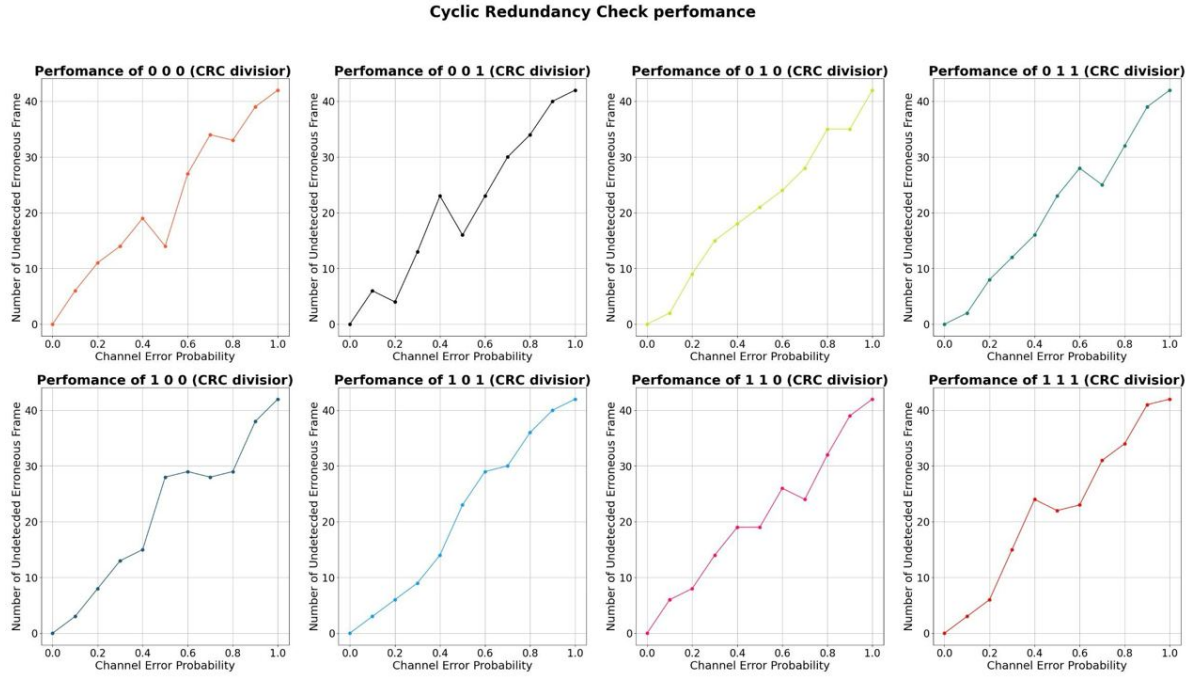
10. But we can see, that number of 1, 2, 3 and 4 errors in message bits larger, than we can see in previous plot. So this algo correctly finds at least one errors in message

3.3.2 Cyclic redundancy check

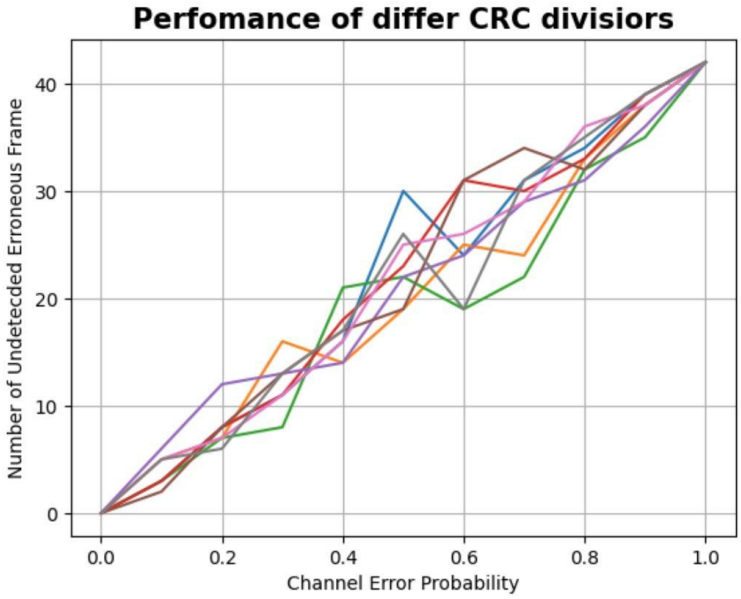
1. In general computation of CRC corresponds to Euclidean division of polinomials
2. $M(x) * x^n = Q(x) * G(x) + R(x)$
3. $M(x)$ - original message polinomial, $G(x)$ - is the degree-n generator polynomial, $R(x)$ - remainder polynomial, $Q(x)$ - quotient polynomial
4. Using modulo operation, it can be stated that $R(x) = M(x) * x^n mod G(x)$
5. In communication, the sender attaches the n bits of R after the original message bits of M, which could be shown to be equivalent to sending out $M(x) * x^n - R(x)$ (codeword)
6. The receiver, knowing $G(x)$ and therefore n, separates M from R and repeats the calculation, verifying that the received and computed R are equal.
7. In practice CRC calculations its long division in binary, except that the subtractions involved do not borrow from more significant digits, and thus become exclusive or operations.
8. In this our custom image we can see how CRC working between sender and receiver sides



9. We also investigated the dependence of the number of errors in the forwarded bit message through the channel with the following probabilities of correcting a bit of information: 0.1, 0.2, 0.3, ..., 1 and for corresponding divisors: 000, 001, 010, 011, 100, 101, 110, 111. The results are shown in the graph below



10. You can see in the graph below that the divisor 111 has the smallest number of errors in the transmitted message, especially when the probability of correcting the information bit on the channel is equal to 0.6



3.3.3 Longitudinal Redundancy Check

1. Longitudinal redundancy check is a form of redundancy check that is applied independently to each of a parallel group of bit streams.

2. The data must be divided into transmission blocks, to which the additional check data is added.

3. LRC differ than VRC exits from the following points

   (a) It stands for Longitudinal Redundancy Check.

   (b) In this redundant row of bits is added to the whole block.

(c) LRC can detect burst errors.

(d) It is also known as 2-D parity checker.

(e) The advantage of using LRC over VRC is that it can check all the burst errors.

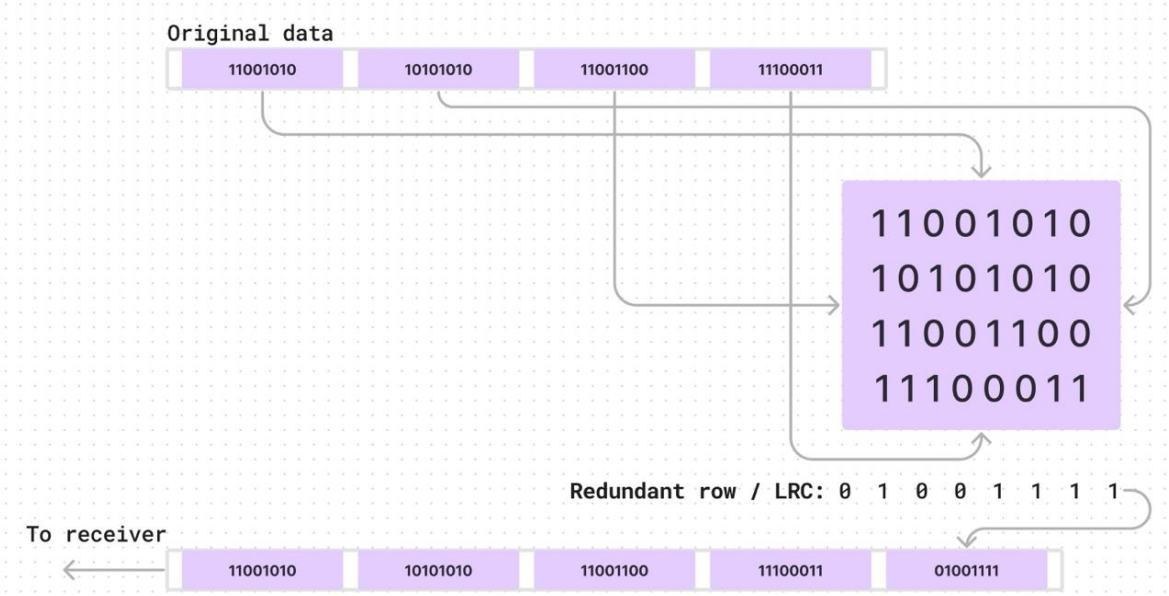
(f) If two bits in data unit is damaged and also in other data unit the same bits are damaged at same position, then it is not capable of detecting such kind of error.

4. LRC pseudocode

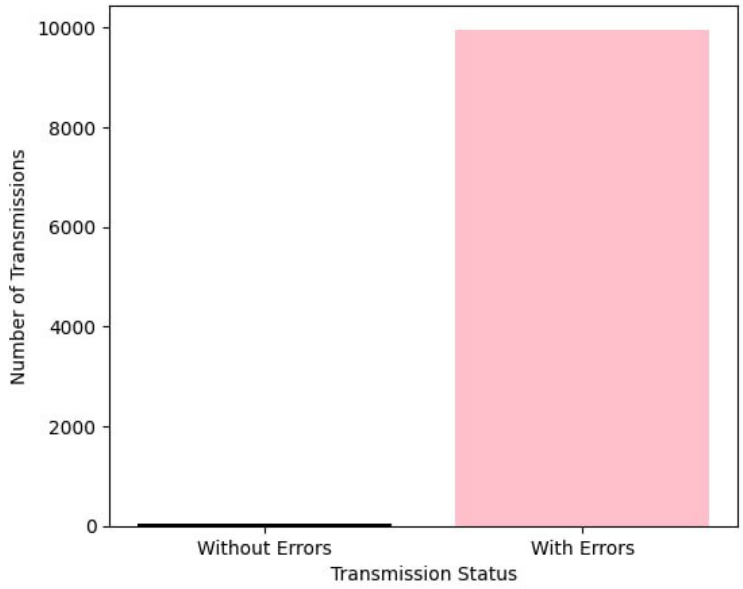
Algorithm 1 LRC pseudocode

$lrc \leftarrow 0$ **for** *each byte b in the buffer* **do**
    end
    $lrc \leftarrow (lrc + b)$ **mod** 256
$lrc \leftarrow (((lrc \oplus 255) + 1)$ **mod** $256)$

5. Here we have depicted how this algorithm works

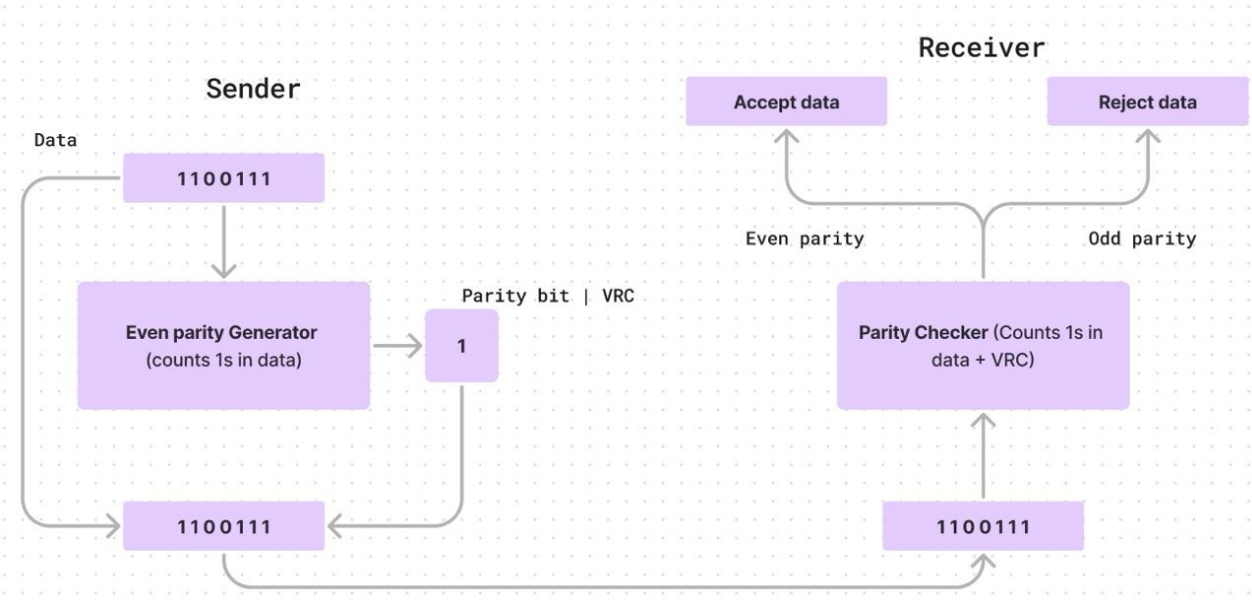


6. As we can see on this plot, it pretty well detects if in block of messages was an error. We tested it on 10000 blocks for 4 messages in each. But this code cannot tell us in which specific message was an error, it can only tell us that it was. Also this code cannot detect if there was an error if error occurred, for example in 4-th and 7-th bit of message 2 and 4-th and 7-th bit of message 3.



3.3.4 Vertical Redundancy Check

1. Vertical Redundancy Check is the error detection method which is used by upper layers to detect error in data
2. It stands for Vertical Redundancy Check
3. In this redundant bit called parity bit is added to each data unit
4. VRC can detect single bit errors
5. It is also known as parity checker
6. The advantage of using VRC is that it can checks all single bit errors but can check odd parity only in the case of change of odd bits.
7. It is not capable of checking the burst error in case of change of bits is even.
8. Here we implemented how this algorithm works



3.3.5 Low density parity-check

1. LDPC(low density parity-check) is a linear error correcting code, which allows to transmit a message over a noisy transmission channel. It encodes messages using parity-chack matrix, simmilarly to Hemmings code. The main difference is in the parity-check matrix itself, which is a large N x M matrix, where N is the length of the codeword (number of bits in the message plus redundant bits) and M is the number of information bits. It's called low-density since each row of the parity-check matrix would ussually contain up to 4 bits, with number of parity checks per column ussually being 3. This allows for more precise error detection and correction, allowing us to correct a larger number of errors.

2. Encoding

   (a) Create an N x M matrix H filled with of zeroes, N being the overall message length, with reduntant bits and M the number of information bits, so the number of check bits is equal to N-M

   (b) For each column in H , three 1's are placed in rows chosen at random, subject only to the constraint that each collumn has to be different.

   (c) The software then runs through the matrix searching for a row with zero 1's or just one 1. If a row has no 1's in it then it is a redundant row. So the software chooses 2 columns in the same row at random and places 1's in those columns. If a row just has one 1 in a row it means that the codeword bit in that column isalways zero. So whenever the software finds a row with just one 1 in it, it randomly picksanother column in the same row andplaces a 1 there.

   (d) The software then calculates the number of 1's per row. Number of 1's per row = (cols x bits per col) / rows.If this is not an integer, the software rounds the value to the next higher integer. If the number of 1's per row, calculated in step 4 is not an integer, it is not possible to have a uniform number of ones in each row.

   (e) The software then runs through the matrix trying to make the number of 1's per row as uniform as possible. For any row (row i, say) containing more number of ones than the value calculated in Step 4, the software picks a column containing a 1 at random and tries to move that 1 to a different row (randomly chosen such that has it a lesser number of ones than the value calculated in step 4) in the same column. The software makes sure that the row so chosen does not have a 1 in that particular column. If the software is not able to find such a row, it just tries with a different column containing a 1 in row i.

3. Decoding

   (a) For transmitted LDPC-encoded codeword $c = c0, c1, \ldots, cn-1$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value

   $$L_{c_i} = \log\left(\frac{P(c_i = 0|channeloutputforc_i)}{P(c_i = 1|channeloutputforc_i)}\right)$$

   .

   In each iteration, the key components of the algorithm are updated based on

these equations:

$$L(r_{ij}) = 2atanh(\prod_{i'V_j \backslash i} tahn(\frac{1}{2}L(q_{i'j})))$$

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i}), \text{ initialized as } L(q_{ij}) = L(c_i)$$

before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

(b) At the end of each iteration, L(Qi) contains the updated estimate of the LLR value for transmitted bit ci. The value L(Qi) is the soft-decision output for ci. If L(Qi) 0, the hard-decision output for ci is 1. Otherwise, the hard-decision output for ci is 0. If decoding is configured to stop when all of the parity checks are satisfied, the algorithm verifies the parity-check equation (H c' = 0) at the end of each iteration. When all of the parity checks are satisfied, or if the maximum number of iterations is reached, decoding stops.

3.3.6 Convolutional codes

1. Encoding

   (a) Select r functions to be applied to the window of length K, for example:

   $$p_0[n] = x[n] \bigotimes x[n-1] \bigotimes x[n-2] p_1[n] = x[n] \bigotimes x[n-1]$$

   (b) Apply this functions to the K bits of message, shift the window 1 to the right. Results of the functions would be our encoded message.

2. Decoding

   (a) The simplest way to decode such massages would be by using a maximum likelihood decoder. It decodes our message as c, the nearest valid codeword with the smallest Hamming distance from the recievest message, finding the maximum likelihood solution.

   (b) let our recieved message be called r. Consider any codeword $\tilde{c}$. If r and $\tilde{c}$ differ in d bits(i.e, their Hamming distance is d), then $P(r|c) = p^d(1-p)^{N-d}$, where N is the length of the recieved word, as well as any valid codeword.

   (c) Take logarithm of this, getting

   $$\log P(r|\tilde{c}) = d \log p + (N-d)\log(1-p) = d \log \frac{d}{1-p} + N \log(1-p).$$

   if $p < \frac{1}{2}$, which is the practical, then $\frac{p}{1-p} < 1$. Overall it comes down to minimizing d, by looking at all the possible close codewords

3.4 Pipeline of the implementation

We want to implement a program where the user will enter the number of the message encoding algorithm, and receive step-by-step instructions on how to encode a message with a particular algorithm, look for errors in it, and how to understand this algorithm from a mathematical point of view, for example, what rules are used in it and with what does it mean In the future, it will be possible to design it in the form of a web program, which will be able to be used by many people who are interested in issues of security and linear algebra.
That is, for the second presentation, we will create the program described above, and for the third, we will design it all in the form of a web site, since it will be convenient for users to use it in this form, and it will be easier to scale it in the future.

3.5 Testing and data

To test different coding algorithms we would write an app which would take user generated message, code it, introduce a selected number of errors into it and then decode it, using our algorithm of choice. Then we would select a number of students who would be asked to tell us the readability of the message. Using this data we would be able to determine the most effective coding algorithm. Also to testing our algorithms we generate random text message convert it to bits and encoding and after some time decoded bits message.

3.6 What we have at that moment

At the moment, we have an implementation of those algorithms that we mentioned above, their research depends on whether they can detect errors, correct them, or simply check whether the data on the receiving end is correct.
We also compiled our algorithms into a Python library that anyone can use because it is open source. **la-coding-theory** package in py-pi `https://pypi.org/project/la-coding-theory/` can be useful for all those people who are interested in coding, working with networks and services and just lovers of linear algebra.

4 References

1. Hamming codes
   `https://users.math.msu.edu/users/halljo/classes/codenotes/Hamming.pdf`
2. Short parity method
   `https://uomustansiriyah.edu.iq/media/lectures/9/9_2020_02_09!08_56_05_PM.pdf`
3. Low-Density Parity Check Codes
   `http://www.jaist.ac.jp/~kurkoski/teaching/portfolio/uec_s05/S05-LDPC%20Lecture%201.pdf`
4. Short Low-Density Parity Check Codes
   `https://web.stanford.edu/~montanar/TEACHING/Stat316/handouts/ldpc.pdf`
5. Data Coding and Error Checking Techniques
   `https://www.accoladeeng.com/documents/Data%20Coding%20and%20Error%20Checking%20Techniques.pdf`

6. Cyclic Redundancy Check
https://perswww.kuleuven.be/~u0068190/Onderwijs/Extra_info/crc.pdf

7. Data correction using Hamming coding and Hash function
https://science.lpnu.ua/sites/default/files/journal-paper/2019/dec/20474/192048-38-42.pdf

8. Convolutional Codes
https://www.cs.princeton.edu/courses/archive/spring18/cos463/lectures/L09-viterbi.pdf

9. VRC codes
https://www.geeksforgeeks.org/vertical-redundancy-check-vrc-or-parity-check/

10. LRC code
https://www.geeksforgeeks.org/longitudinal-redundancy-check-lrc-2-d-parity-check/

11. CRC perfomance
https://www.researchgate.net/publication/342448331_Performance_Analysis_of_Cyclic_Redundancy_Check_CRC_Error_detection_Technique_in_the_Wireless_Sensor_Network

12. LDPC codes
https://www.accelercomm.com/overview-low-density-parity-check#:~:text=Low%20%2D%20density%20parity%20check%20(LDPC)%20code%20is%20a%20linear,transmitted%20via%20very%20noisy%20channels.

13. Error detection codes
https://www.tutorialspoint.com/what-are-error-detecting-codes