Error Correction Codes Assignment

Polydoros Giannouris - 9746
Electrical and Computer Engineering
Aristotle University of Thessaloniki
Thessaloniki, Greece
polydoros@ece.auth.gr

Anestis Kaimakamidis - 9627
Electrical and Computer Engineering
Aristotle University of Thessaloniki
Thessaloniki, Greece
anestisk@ece.auth.gr

I. Introduction

This assignment's aim is to study error correction codes in different channels. In the first part we will study the binary symmetric channel where we will compare multiple linear codes with respect to their complexity, transmission and error rate. Next we will study Low Density Parity Check (LDPC) codes in the binary erasure channel. For the irregular LDPC code we will test dual optimization problems for parameter selection. Afterwards we will study the resulting code with respect to its complexity, use of channel capacity and error correcting ability. Finally we will compare irregular and regular LDPC codes, showcasing their differences. Code can be found on https://github.com/PolydorosG/Error-Correction-Codes.

II. Binary Symmetric Channel

Throughout this assignment we will assume that the messages belong to a Galois Field and in particular GF(2). Therefore messages consist of bits valued 0 or 1. In the binary symmetric channel (BSC) the transmitter wishes to send a bit to the receiver. Due to noise the bit will be ”flipped” with a probability of p, otherwise it will be received correctly. Figure. 1 shows the process of a bit travelling through a binary symmetric channel.

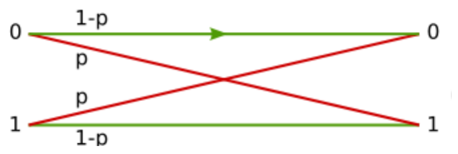

Figure 1. The binary symmetric channel. Green vertices denote no error while red denote ”flipping”.

A. Repetition code

As a baseline code we implement Repetition Coding. Particularly each bit of the message to be transmitted is repeated r times. The code word is then decoded by a majority vote of each r consecutive bits. We notice that although increasing r tends to decrease the bit error rate, it comes at a great cost for the transmission rate.

1) Complexity: Assuming the message consists of m bits and each bit is repeated r times, encoder complexity is $O(r * m)$ write operations and $O(m)$ read operations, as it consists of 2 nested for loops. Our experimental results support the theoretical assumptions for both m (Figure. 2) and r (Figure. 3).

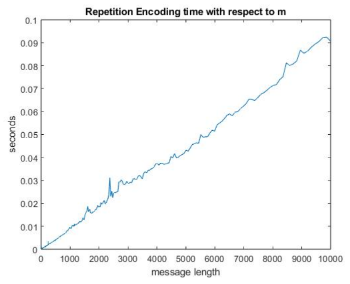

Figure 2. Repetition encoding time for constant r and variable m.

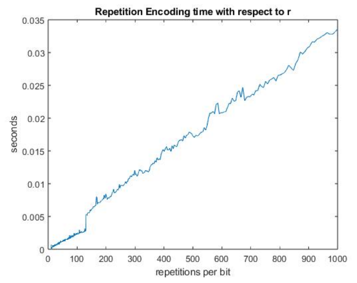

Figure 3. Repetition encoding time for constant m and variable r.

As the decoder follows the same process in reverse, the complexity is the same as the encoder. Once again the experimental results support our theoretical computations (Figures 4, 5).

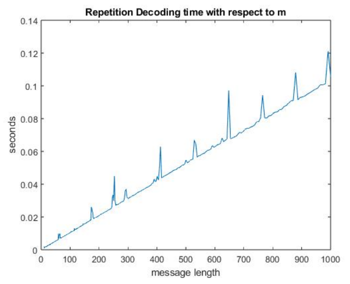

Figure 4. Repetition decoding time for constant r and variable m.

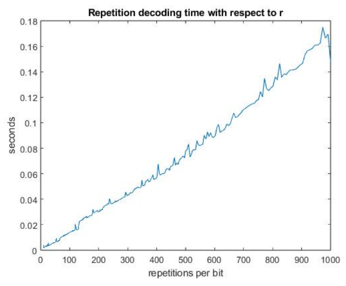

Figure 5. Repetition decoding time for constant m and variable r.

2) Transmission Rate: The transmission rate of the repetition code can be calculated as:

$$\rho = \frac{1}{r} \tag{1}$$

3) Error Probability: When it comes to error probability it is obvious that more repetitions lead to better error correcting, however the gain diminishes as r increases (Figure. 6). Moreover as $\rho = \frac{1}{r}$, the loss in transmission rate is significant.

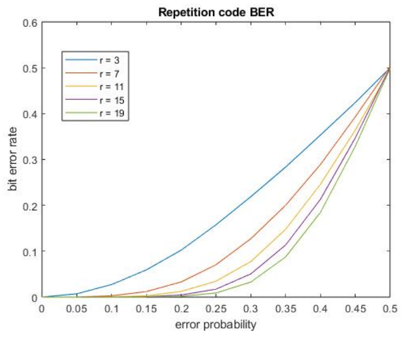

Figure 6. Bit error rate of repetition code for 5 different r values.

B. Hamming Code

As an alternative we implement the Hamming code. In the Hamming code, parity bits are chosen so that one error can be corrected. In this assignment we implement a function to calculate the parity check (H) and generator matrix (G) for any pair (n, k).

1) Complexity: For the Hamming (n, k) code, our implementation uses a generator matrix to encode the message. Thus encoding consists of a matrix-vector multiplication meaning encoding complexity is $O(k*n)$. We note that k and n are not independent thus our experiments will consists of increasing them both at the same time and plotting against $(k * n)$. Plotting the elapsed time of encoding against $(k * n)$ verifies our assessment (Figure. 7). We believe the irregularity for low values of $(k * n)$ stems from software memory management.

Similarly decoding the message consists of multiplying with the parity check matrix and flipping bits according to the syndrome found. Thus complexity is $O((n-k)*n)$. Figure. 8 verifies the calculated complexity, while presenting the same problem as Figure. 7.

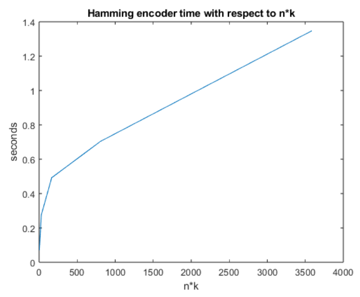

Figure 7. Hamming encoding time with respect to $k * n$.

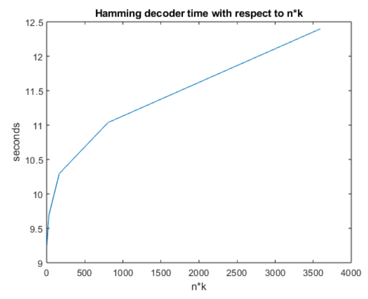

Figure 8. Hamming decoding time with respect to $k * n$.

2) Transmission Rate: We notice that in order to achieve higher rates we are forces to increase the total bits while still only being able to correct one error as can be seen on Table I.

Total bits	Parity bits	Data bits	Transmission rate
3	2	1	0.33
7	3	4	0.571
15	4	11	0.733
31	5	26	0.839
63	6	57	0.905

Table I
HAMMING CODE RATE.

3) Error Correction: The Hamming code is designed to correct up to one error in the input sequence. Thus the only gain from increasing code word length n is the transmission rate as can be seen on Table I. Figure. 9 shows the BER of 5 different Hamming codes. We notice that the smaller the code word, the more errors we are able to correct. However as error probability exceeds a certain threshold, the corrections being made are wrong themselves, equalizing the performance of all presented codes. For any practical use of the Hamming code, estimation of the channel's error probability is critical, as transmission rate differences between Hamming codes are significant. Another helpful quality of Hamming codes that has not been investigated in this assignment is their ability to detect one and two bit errors, allowing us to resend a message if error is detected but not corrected.

III. Binary Erasure Channel

In the binary erasure channel the transmitter wishes to send a bit to the receiver. Due to noise the bit will be "lost" with a

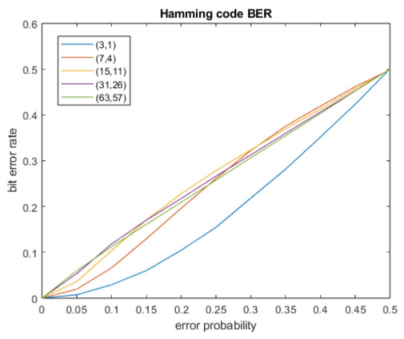

Figure 9. Bit error rate of 5 different Hamming codes.

probability of p, otherwise it will be received correctly. Note that no bit "flipping" occurs in this channel, thus if a bit is received we are certain it is correct. Figure. 10 shows the process of a bit travelling through a binary erasure channel.

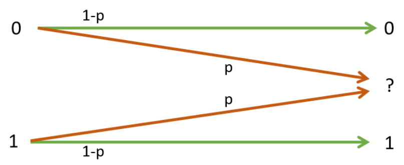

Figure 10. The binary erasure channel. Green vertices denote no error while red denote loss of bit.

IV. Irregular LDPC Optimization

For the irregular LDPC code, since variable and check nodes have different degrees, we start by iteratively running the dual optimization problem specified on page 40 of our notes (function optimize_connectivity_nodes). Although ideal, this process returns a code length n that is not fit for any practical use, since we would have to send data in huge batches. In order to combat this problem we specify n (function ldpc_set_n). Setting n allows us to combine it with the ideal degrees found previously into the closest possible approximation (notes page 48).

A. Parity Check Matrix

Given the degrees returned from the previous step we have to construct the parity check matrix H of size $(n-k) \times n$, where k is the number of check nodes. As instructed this was done randomly, by assigning vertices to check nodes until their degree is met. Even for large codes this method misses few vertices, which is to be expected. We note that missed vertices are usually as few as 5.

B. Encoding process

As no encoding process was recommended we developed an iterative process of testing parity bits until all equations are met. This works well for smaller codes but is very slow for longer ones. A script using the code can be found in our deliverable folder despite not being used in the experiments. Without loss of generality we decided to use all zero code words, since parity bits will by default satisfy all equations if equal to zero as well.

C. Decoding Process

The decoding process utilises the belief propagation algorithm. Essentially the idea of the algorithm is iterating through check nodes. For every check node, the algorithm checks if only one of its incoming bits is erased, allowing it to correct it by summing over all known bits. This is equivalent to solving an equation with one unknown variable. Each variable node correction directly affects the equations of other check nodes, allowing us to repeat the process until convergence to the sent message.

D. Results

1) Complexity: The complexity of the LDPC decoder can be calculated in terms of operations. Our implementations complexity is $O((n-k)*n)$ per iteration. As we force algorithm to stop after $max\_iterations$ regardless of convergence the total worst-case complexity is $O(max\_iterations*(n-k)*n)$.

2) Decoder Convergence: As the complexity mainly stems from the decoder, the maximum allowed number of iterations greatly affects decoding time. For this reason we compare the bit error rate of the same code, while allowing a different number of iterations each time. It is obvious that the number of iterations is critical to the bit error rate. This proves the aforementioned statement about the belief propagation algorithm, that the correction of variable values allows other check node equations to be solved.

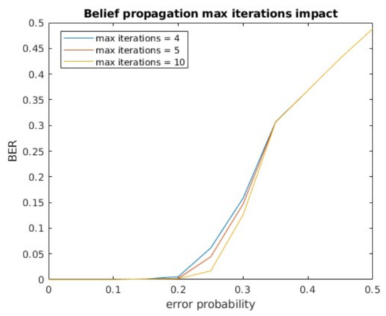

Figure 11. Bit error rate with respect to error probability for 3 different maximum iterations ($n=1000$, $d_c^{max}=20$, $d_v^{max}=20$, $e_{code}=0.3$).

3) Error Design and Code Rate: We also show the importance of specifying the expected channel error when designing an irregular LDPC code. It is clear that choosing a higher erasure probability leads to better error corrections (Figure. 12), however as can be seen on Table II that comes at the cost of code rate. Lastly even if a code is designed for erasure probability e, errors tend to occur at a lower erasure probability e_{BP}.

For reference we note that while the theoretical Shannon capacity is 0.23 the code designed for the correct error probability 0.3 is able to accurately transmit information for erasure probability $e < 0.18$.

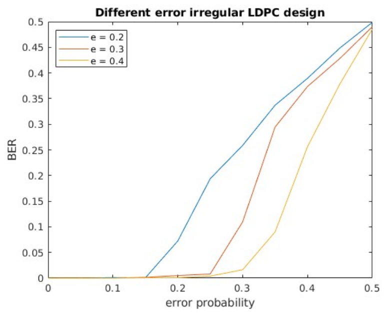

Figure 12. Bit error rate with respect to error probability for 3 different maximum iterations ($n = 1000$, $d_c^{max} = 20$, $d_v^{max} = 20$.

Design error	ρ	e_{Sha}	e_{BP}
0.2	0.76	0.24	0.13
0.3	0.67	0.23	0.18
0.4	0.58	0.42	0.25

Table II
IDEAL VS IRREGULAR LDPC RELIABLE CORRECTIONS.

V. REGULAR LDPC

The regular LDPC differs from the irregular in that all variable nodes have the same degree and so do the check nodes. However the same properties apply when it comes to encoding and decoding given the parity check matrix, so our previous code is applicable here.

A. Parity Check Matrix

In order to construct the parity check matrix we use Gallager's construction as described in [1]. The code generates a basic sub-matrix by filling each row with as many ones as the degree of each check node and zeros elsewhere. It then randomly permutes the columns of the sub-matrix to generate multiple sub-matrices. These sub-matrices are concatenated to form a larger matrix, which is then repeated to generate the final parity check matrix. Our code is largely inspired by [2].

B. Regular vs Irregular code

In this section we will study the differences of regular and irregular codes. Although regular codes are much easier to design, it has been shown that their irregular counterpart is able to approach the Shannon limit much better. Our experiment consists of comparing an irregular LDPC code designed for erasure probability $e = 0.35$, maximum allowed degree for both variable and check nodes $max_v = max_c = 20$, letting the optimization problem decide the ideal values. The irregular LDPC code generated has 3006 number of edges and code rate 0.63. As far as the regular LDPC code is concerned, it was designed with variable degree $d_v = 4$, check degree $d_c = 10$, resulting in 0.6 code rate. The number of edges here is 4000. Both regular and irregular codes are specified a code word length of 1000.

As Figure. 13 indicates, the irregular LDPC achieves lower BER than the regular LDPC with similar code rates, indicating operation closer to the channel's Shannon capacity.

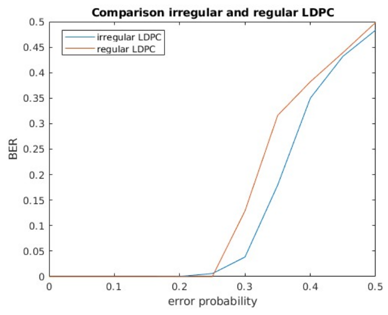

Figure 13. Bit error rate with respect to error probability for a regular and an irregular LDPC code with code rates $r = 0.6$ and $r = 0.63$ respectively.

As a final experiment we try to find the amount of reduction to e_{Sha} of the regular LDPC that would have a similar performance to the irregular.

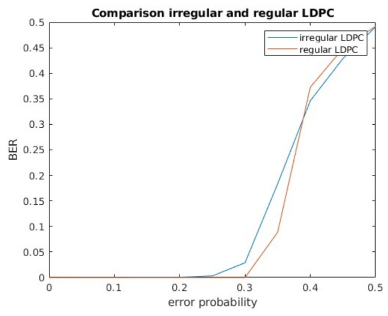

Figure 14. Bit error rate with respect to error probability for a regular and an irregular LDPC code with code rates $r = 0.4$ and $r = 0.63$ respectively.

Lastly comparing Table III and Figure. 14 shows that although error onset may be quicker for this particular irregular LDPC code, the number of edges is lower and, more significantly, the irregular LDPC code is able to correct significantly more errors once they appear. We also comment that these plots are not deterministic due to the nature of the channel. Thus small deviations are possible, that would explain why errors appear first on the irregular LDPC.

Code	ρ	e_{Sha}	e_{BP}
Irregular LDPC	0.63	0.27	0.23
Regular LDPC	0.60	0.40	0.24
Regular LDPC 2	0.40	0.60	0.3

Table III
IDEAL VS IRREGULAR LDPC RELIABLE CORRECTIONS.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," in IRE Transactions on Information Theory, vol. 8, no. 1, pp. 21-28, January 1962, doi: 10.1109/TIT.1962.1057683.
[2] S. Kalamkar. "Gallager's Construction of Parity Check Matrix for LDPC Codes." MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/44454-gallager-s-construction-of-parity-check-matrix-for-ldpc-codes (accessed Feb. 7, 2023).