Adaptive message-passing decoding: ways to implement and limits of applicability

Research Seminar Project MTAT.07.022
Author: Janno Siim
Advisors: Vitaly Skachek, Yauhen Yakimenka
University of Tartu

In this report we study the decoding algorithms for a family of codes called LDPC codes used over the binary erasure channel (BEC). LDPC codes are able to work at rates near Shannon capacity with linear time decoding algorithms. Those linear time decoding algorithms sometimes fail even when Gaussian elimination would still be able to restore the codeword. In this report we try to experimentally test a novel way of decoding LDPC codes over BEC and to analyse its performance.

Introduction

First we define some basic definitions of coding theory.

Definition 1. *An (n, M)-code C over an alphabet F is a set of size $M > 0$ containing vectors (called codewords) over F of length n.*

Definition 2. *Let $\mathbb{F}$ be a finite field. An (n, M)-code C over $\mathbb{F}$ is called linear if C is a nonempty linear vector subspace of $\mathbb{F}^n$.*

Definition 3. *Channel is a triplet $(\Sigma_{in}, \Sigma_{out}, Prob)$ where Σ_{in} is the input alphabet, Σ_{out} is the output alphabet and $Prob : \Sigma_{in} \times \Sigma_{out} \to [0, 1]$ is the probability function.*

Definition 4. *Binary erasure channel (BEC) with probability $p \in [0, 1]$ is a channel with input alphabet $\{0, 1\}$, output alphabet $\{0, 1, \epsilon\}$ and probability function $Prob : \{0, 1\} \times \{0, 1, \epsilon\} \to [0, 1]$ defined as*

$$Prob(a, b) = \begin{cases} p & \textit{if } b = \epsilon \\ 1 - p & \textit{if } a = b \\ 0 & \textit{otherwise} \end{cases}$$

BEC is the channel model that we are going to use for the rest of the report.

Let C be a binary $[n, k, d]$ linear code, where n is the length, k is the dimension, and d is the minimum Hamming distance between any two codewords in C.

Definition 5. *Parity-check matrix of a linear code C over field $\mathbb{F}$ is $r \times n$ matrix H such that*

$$\bar{c} \in C \Leftrightarrow H\bar{c}^T = \bar{0}^T$$

where $\bar{0}$ is a zero vector of length r.

Low-density parity-check (LDPC) code is a linear code that uses a sparse parity-check matrix, i.e. only a small portion of it is filled by nonzero elements.

There are two general types of LDPC codes: regular and irregular. Difference lies in how the nonzero elements are distributed in the parity-check matrix. In this report we mostly deal with binary regular LDPC codes over BEC.

Definition 6. *We call a linear code C a (c, d)-regular LDPC code if every row in a parity-check matrix of C has d nonzero elements and every column has c nonzero elements. [Gallager, 1962]*

LDPC codes were first discovered in 1962 by Robert Gallager. Then deemed to be impractical and forgotten until the second half of the 90's. As of today topic of LDPC codes is an active area of research. LDPC codes are used in several practical application like 10GBASE-T Ethernet standard and many others. [Cevrero, Leblebici, Ienne, Burg, 2010]

LDPC codes are able to reach the Shannon's limit i.e. the code rate can be as high as theoretically possible while still being able to very efficiently decode the received codeword with high probability. Until the discovery of Turbo codes in the beginning of 90's such code rates where unheard of. Overview of the history and importance of LDPC codes can be seen at MIT's newsoffice webpage. [Hardesty, 2010]

Definition 7. *A maximum-likelihood (ML) decoder for the code C is a function $D : \Sigma_{out}^n \to C$ defined as*

$$D(\bar{y}) = \bar{z} \textit{ such that}$$

$$\bar{z} = \arg\max_{\bar{c}\in C} \Pr(\bar{y} \textit{ received} \mid \bar{c} \textit{ transmitted}).$$

Another important feature of LDPC codes is that they can be decoded with iterative decoding algorithms that work in linear time. These algorithms work well in practise but they are still suboptimal in a sense that they may not be able to recover codeword even when it would be possible to do so with ML decoding. ML decoding in the context of BEC means solving a linear system of equations.

We would like to reduce the gap between iterative decoding and ML decoding while still being close to linear time decoding. It is known that recovering as many bits as possible with iterative decoding and then finding the remaining bits by solving the linear system of equations by Gaussian elimination is not fast enough to be practical.

Reason for iterative decoder failures on BEC is well understood. These failures are caused by structures called stopping sets in the parity-check matrix. Adaptive removal of stopping sets in order to allow iterative decoder continue its work is a possible improvement.

In this report we study how does speed of decoding change if we use iterative decoding together with a methods that we shall call two elimination and three elimination which remove the stopping sets whenever possible and then uses Gaussian elimination on top of that. Using two elimination and three elimination can lead to more efficient decoding.

Chapter 1 covers an algorithm for generating parity-check matrix of an LDPC code and the decoding algorithms that we are going to test. Chapter 2 talks about the implementation of the algorithms and results of the test. Chapter 3 contains a novel decoding algorithm - three elimination.

1 Decoding LDPC Codes

In this chapter we define a decoding algorithms, which we study in Chapter 2.

Let $\bar{y} = (y_1, y_2, \ldots, y_n)$ be the binary word that is received from the BEC and H be an $r \times n$ parity-check matrix of a linear code over $\mathbb{F}_2$. Let $\bar{x} = (y_{\alpha_1}, y_{\alpha_2}, \ldots, y_{\alpha_q})$ be a vector of erased bits.

We know that

$$H\bar{y}^T = \bar{0}^T$$

where $\bar{0}$ is a zero vector of length r. Some of the bits in $\bar{y}$ are known, because they are not erased in the course of transmission. We can simplify the previous equation to

$$H'\bar{x}^T = \bar{b}^T$$

$$H' \begin{pmatrix} y_{\alpha_1} \\ y_{\alpha_2} \\ \vdots \\ y_{\alpha_q} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix}.$$

Matrix H' here is a submatrix of H consisting only of the columns of H corresponding to the unknown bits in the received word $\bar{y}$ and $\bar{b}$ is the sum of the columns of H corresponding to positions of ones in $\bar{y}$. Our goal in decoding is to recover the vector $\bar{x}$.

First we discuss algorithms for constructing parity-check matrices for LDPC codes and data structures used to store them.

Data Structure For Sparse Matrices

Let us define the data structures that can be used for storing the parity-check matrix of LDPC codes. Classical way of storing a matrix is by using a two dimensional array. In the case of sparse matrices it is impractical because most of the matrix consists of zeros. It is more memory-efficient to store the nonzero positions only. Additionally algorithms can work faster by taking advantage of the sparse structure of a matrix.

Several ways of storing sparse matrices exist but for iterative decoding algorithm the following structure is especially convenient.

Let H be a sparse $r \times n$ matrix over the field $\mathbb{F}_2$. Matrix H will be stored as two arrays of lists

$$rows_i = \{j \mid H_{ij} = 1\} \text{ for } i = 1, 2, \ldots, r$$

$$cols_j = \{i \mid H_{ij} = 1\} \text{ for } j = 1, 2, \ldots, n.$$

Notice that storing the row lists alone would already contain all the information about the structure of the matrix. Reason for storing the column lists in addition to row lists, is that then we have faster look up access to a specific column of a matrix.

Parity-check Matrix Generation

We describe one simple way of generating parity-check matrices for LDPC codes.

To create a parity-check matrix for (c, d)-regular LDPC code of size $mc \times md$ where $m \in \{1, 2 \dots\}$ we do the following. First we write an $m \times md$ matrix as a matrix of m blocks with size $m \times d$.

$$\left(\begin{array}{cccc|cccc|c|cccc} 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & & & & & & & & & & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & \dots & 1 & 1 & \dots & 1 \end{array}\right)$$

This will be the first block of parity-check matrix H. All the other $c - 1$ blocks below the first block will be a random column permutations of this first block.

Example 1. For $c = 3, d = 4$ and $m = 3$ the matrix can take a form

$$H = \left(\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right).$$

Gaussian Elimination

Gaussian elimination is an algorithm that can be used for solving a linear system of equations.

The main idea is to use elementary row operation on the matrix $\left(H' \mid \bar{b}^T\right)$ to transform it into the so-called reduced row-echelon form. Reduced row-echelon form means that matrix is an upper-triangle matrix where each row starts with a one and the first one in every row is to the right of the first one of the row above it.

Example 2. A binary matrix in reduced row-echelon form:

$$\left(\begin{array}{ccccc|c} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Only if the number of nonzero rows in row-echelon form is equal to the number of variables, has the system a unique solution. If that is the case then it is easy to substitute the variables back starting from the lowest nonzero row.

Gaussian elimination has the time complexity $O(n^3)$. LDPC codes use parity-check matrices over the field $\mathbb{F}_2$ which mostly contain zeros. There exist more efficient methods for solving linear systems of this type than the generic Gaussian elimination [Burshtein, Miller, 2004]. Still even the more advanced methods are not practical to be used on their own due to high time complexity.

Iterative Decoding

We are going to look at a method of decoding that works significantly faster than Gaussian elimination. This method can however fail even when a unique solution exists.

LDPC codes can be decoded using iterative decoding algorithms. Sometimes these algorithms are also called message-passing algorithms because they can be viewed as sending messages over the edges of a certain type of bipartite graph called a Tanner graph. We ignore the graph viewpoint and use the sparse matrix data structures that were previously described.

Basic idea of iterative decoding is that we start to go iteratively over the rows of the matrix H'. If we see a row of weight 1 (only one nonzero element) then it is possible to find the value of the corresponding missing bit.

Suppose the i-th row in the matrix H' has weight 1 and the only nonzero element is at the j-th column. Then the equation view of that row will be

$$0 \cdot y_{\alpha_1} + \dots + 0 \cdot y_{\alpha_{j-1}} + 1 \cdot y_{\alpha_j} + 0 \cdot y_{\alpha_{j+1}} + \dots + 0 \cdot y_{\alpha_q} = b_i$$

$$y_{\alpha_j} = b_i.$$

We have successfully recovered the missing bit y_{α_j}. Because we know the codeword bit corresponding to the column j in the matrix H', we can remove this column. Before we do that we also must update the vector $\bar{b}$ to be $\bar{b}^T + b_i H'_j$ were H'_j is the j-th column of the matrix H'. After removing the j-th column from matrix H' then the i-th row will only consist of zeros and therefore we can remove it as well. By performing these steps, we obtain a new matrix H'.

After one such removal is done, we have a new matrix H' and we can continue iterating over the rows to look for the next row of weight one and then repeat the removal process. Also because we have a sparse parity-check matrix then we would expect it to have quite many rows of weight one.

Algorithm 1 Iterative decoding algorithm

```
Require: $r, n, cols, rows, \bar{y} \in \{0, 1, \epsilon\}^n$
 1: $stuck \leftarrow false$
 2: while not $stuck$ do
 3:     $stuck \leftarrow true$
 4:     for $i \leftarrow 1$ to $r$ do
 5:         if $(\sum_{j \in rows_i} I(y_j = \epsilon)) = 1$ then
 6:             $j_0 \leftarrow j \in rows_i$ such that $y_j = \epsilon$
 7:             $y_{j_0} \leftarrow \bigoplus_{j \in rows_i, y_j \neq \epsilon} y_j$
 8:             $stuck \leftarrow false$
 9:         end if
10:     end for
11: end while
12: return $\bar{y}$
```

Algorithm 1 shows the precise description of the iterative decoding algorithm for the sparse matrix data structure where r is the number of rows, n is the number of columns, $cols$ and $rows$ form the parity-check matrix described by the sparse matrix data structure and $\bar{y}$ is the received word. I is a function that outputs 1 if the input is true and 0 otherwise. Symbol $\bigoplus$ denotes XOR operation.

In line 5 in the Algorithm 1 we check if row has weight one. In line 6 we find the position of the corresponding erased bit. In line 7 we fix the erased value.

In some cases iterative decoder fails to find the solution for the system of equations. It is relatively easy to see when this happens.

Definition 8. *Let matrix H be a parity-check matrix of a linear code. We call a set of column vectors of matrix H a stopping set if the submatrix formed by those columns does not contain a row of weight one.*

If positions that are missing from the received word $\bar{y}$ correspond to the columns of the stopping set then the iterative decoder will stop.

One possible thing to do after the iterative decoder has failed, is to continue solving the system of equations with Gaussian elimination. Figure 1 describes this model of decoding. Gaussian elimination is relatively slow however, as it takes $O(n^3)$ time where n is the upper bound on the number of equations and unknowns.

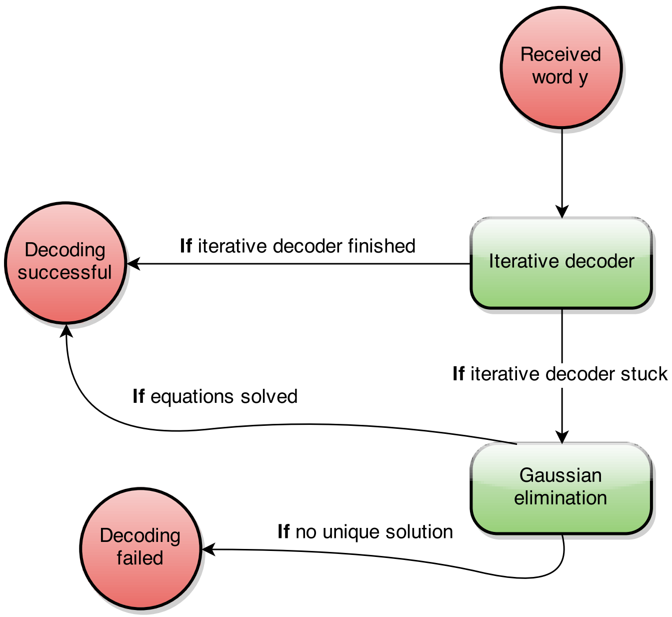

Figure 1: *Decoding model where iterative decoder is used together with Gaussian elimination.*

Two elimination

It would be more time efficient if the iterative decoder would recover as much of the bits as possible and by the time we have to use Gaussian elimination only a very small fraction of the erasures would still be remaining. Could we somehow prolong the iterative decoder phase of the model in Figure 1?

One such technique has been proposed for doing this in [Olmos, Murillo-Fuentes, Perez-Cruz, 2012].

Let us look at a row of weight two. Let the i-th row in the matrix H' have weight two and let j_1 and j_2 be the positions of ones in that row. Corresponding equations will be

$$y_{\alpha_{j_1}} + y_{\alpha_{j_2}} = b_i$$

If $b_i = 0$ then $y_{\alpha_{j_1}} = y_{\alpha_{j_2}}$ and if $b_i = 1$ then $y_{\alpha_{j_1}} = y_{\alpha_{j_2}} + 1$. Knowing $y_{\alpha_{j_1}}$ will immediately reveal $y_{\alpha_{j_2}}$ and vice-versa. This means that we could remove one of the variables. We can simply substitute $y_{\alpha_{j_1}}$ in the equations with $y_{\alpha_{j_2}} + 1$.

In the matrix terms it could be done as follows. We can remove the i-th row. If $b_i = 1$ then we must flip the bits in the vector $\bar{b}$ that correspond to the positions of ones in the j_1-st column. Then we can add j_1-st column to j_2-nd column and after that we can remove j_1-st column. We will refer to this method as two elimination from now on.

When the iterative decoder gets stuck, we can try to find a row of weight two and remove it as described before. After that we might have removed the stopping set and the iterative decoder can continue its work. Otherwise we can try to remove even more rows of weight two. Only when there are no rows of weight two left we turn to Gaussian elimination. Figure 2 describes this model of decoding.

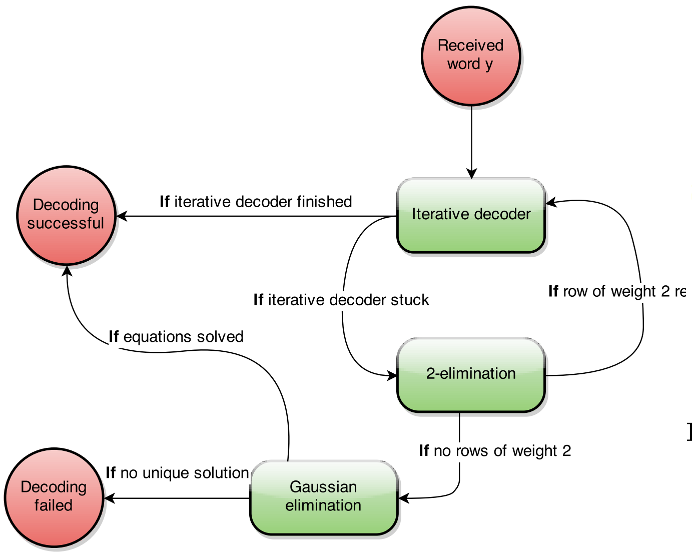

Figure 2: *Decoding model from figure 1 with addition of two elimination.*

This model of decoding could potentially work faster than the one that is described in Figure 1 but this should be experimentally verified. This is what we do in the next chapter.

2 Experiments

Implementation

In order to test the effectiveness of two elimination, all the algorithms in the previous chapter were implemented. For convenience the algorithms are implemented in python 2. Code is available from the following repository: `https://bitbucket.org/JannoSiim/ldpc_experiment`.

Test Results

We compare two elimination decoding model (Figure 2) with iterative decoding and Gaussian elimination. Properties of error correction efficiency and speed are considered.

Definition 9. *Let n be the length of the code and r the number of bits that were not recovered by the decoder. We call the value $\frac{r}{n}$ the bit error rate.*

First test measures the dependency between erasure probability and bit error rate. Both this and the second test use $(3, 6)$-regular LDPC codes. For this test the code length of 1800 is used. Results can be seen on Figure 3.

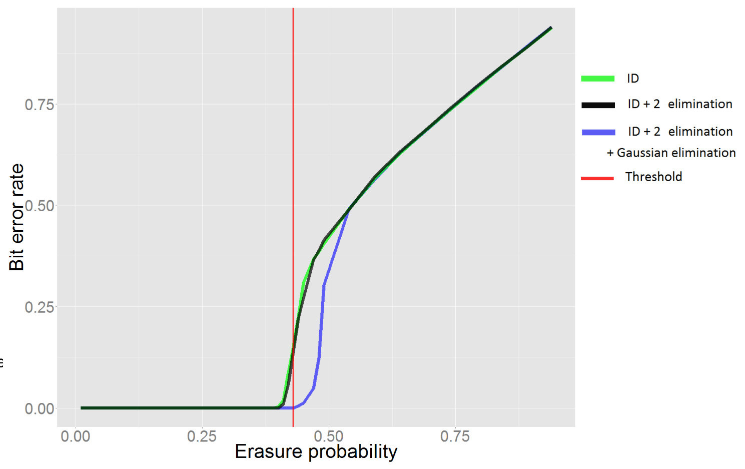

Figure 3: *Results of the bit error rate test. ID in the legend stands for iterative decoder. For LDPC codes of sufficiently high length iterative decoder can have bit error rate close to zero, up to the threshold line.*

Figure 3 shows that only a small decrease in bit error rate happens if two elimination is combined with iterative decoder compared to using iterative decoder alone.

Secondly we look how the speed of decoding depends on the code length. Again $(3, 6)$-regular LDPC codes are used. Results can be seen on Figure 4.

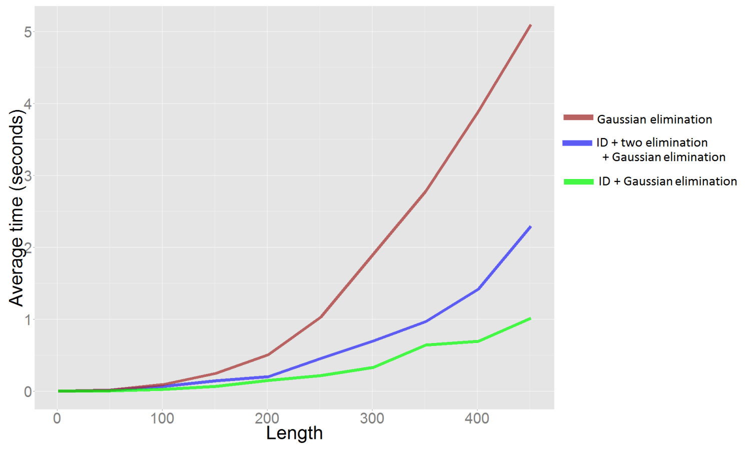

Figure 4: *Results of the speed measurements. ID in the legend stands for iterative decoder.*

In the current implementation two elimination step in the decoding seems to make the decoding slower instead of making it faster. Two elimination step is adding overhead to decoding but giving only a minor decrease in the bit error rate.

First results of the adaptive decoding algorithm seem to imply that it does not offer any improvement compared to iterative decoder combined with Gaussian elimination. Final conclusion about the applicability cannot be made however. There are many possible places for errors in the implementation and the testing procedure. It should be carefully verified that implementation is working correctly and efficiently. In addition it is possible that decoding behaviour changes for codes of greater length.

3 Three Elimination

Given that eliminating rows of weight two does not give sufficient improvement in bit error rate then maybe removal of rows with weight three does it. Looking at distributions of row weights in cases where two elimination is not successful shows that rows of weight three are on average much more frequent than rows with higher weight.

Three elimination could be done as follows:

1. Run two elimination algorithm. If codeword is fixed then exit, otherwise go to step 2.
2. Find a row of weight three according to some rule. Lets denote the row in equations form as $x + y + z = c$ where x, y, z are variable and $c \in \{0, 1\}$ is a constant. If there are no rows of weight three abort the algorithm (or do Gaussian elimination).
3. Choose according to some rule one of the three variables x, y, z for substituting into other equations. For simplicity let that variable be x.
4. Substitute $x = y + z + c$ into all the equations that contain x. Go back to step 1.

Depending on the way the rows of weight three in step 2 and variable in step 3 are picked, it can give different bit error rate results.

Naive Three Elimination

Let us first study one very simple way of picking rows and variables. We iterate over the rows and pick the first one that has weight three. From that row we pick the first variable for substitution. Let us call this algorithm naive three elimination.

We compare bit error rate of naive three elimination to iterative decoding, two elimination and Gaussian elimination. We restrict erasure probability to the range 0.35 to 0.55 because that is the region where the algorithms differ. Each data point is averaged over 200 iterations. Figures 5 and 6 show the results of the experiment.

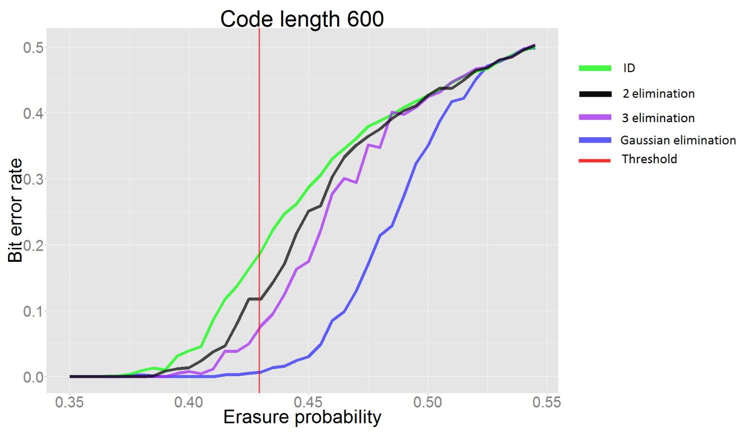

Figure 5: *Bit error rate of naive 3 elimination compared to iterative decoding, 2 elimination and Gaussian elimination with code length 600.*

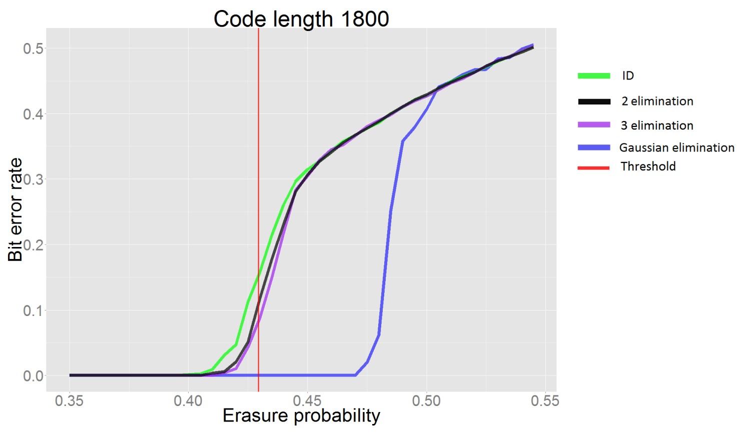

Figure 6: *Bit error rate of naive 3 elimination compared to iterative decoding, 2 elimination and Gaussian elimination with code length 1800.*

For smaller code length naive three elimination seems to give a significant advantage in bit error rate but as the length of the code increases that advantages diminishes.

Other Row Picking Strategies

Next we test if picking equations and variables for substitution in three elimination in some other way gives a greater advantage. We consider following strategies:

Naive Pick first row of weight three and from that row the first variable for substitution (as was done before). This is on the average the fastest strategies of the four.

Random Pick a random row out of all the rows of weight three and from that row pick a random variable out of three possible variables. This strategy needs that we iterate once over all the rows to find those that have weight three.

First reduction Iterate over the rows and pick the first row and variable that reduces at least one other row weight.

Best score For each row r of weight three and variable x from that row calculate $Score(r, x) := n_{R2}(r, x)\cdot s_{R2}+n_R(r, x)\cdot s_R-n_I(r, x)\cdot s_I$. We look how the substitution of the variable x according to row r would change the weight distribution of rows. Here $n_{R2}(r, x)$ is the number of rows reduced to weight two, $n_R(r, x)$ is the number of rows reduced to higher weight than two and $n_I(r, x)$ is the number of rows that increase by the substitution. Values s_{R2}, s_R and s_I are nonnegative constants that give score to each row of the corresponding type. Rows that would get reduced contribute nonnegative values to the score and rows that would increase in weight contribute nonpositive values to the score. We pick the row of weight three with the highest score.

We compared the four strategies of picking rows and variables on codes of length 900. Results can be seen on figure 7. There does not seems to be a significant difference between bit error rate of the strategies. For best score strategy constants $s_{R2} = 10$, $s_R = 1$ and $s_I = -0.1$ are used. Some other constants were tested as well but they did not seem to change the end result very much.

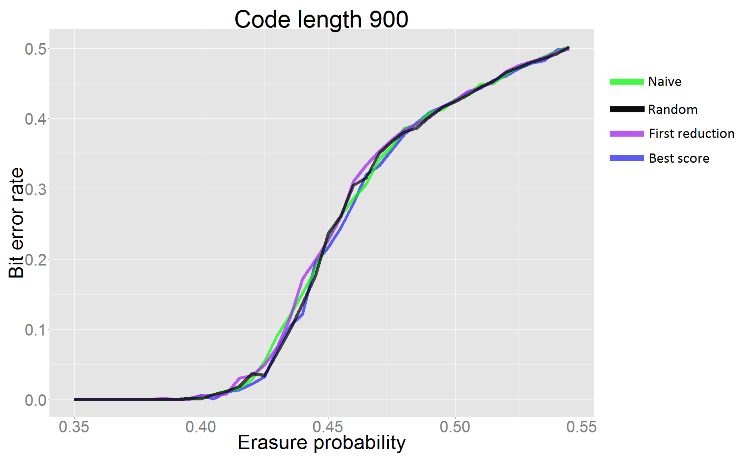

Figure 7: *Comparison of different row and variable picking strategies for three elimination.*

Conclusion

In this report we studied decoding algorithms for LDPC codes over binary erasure channel. LDPC codes can be decoded with linear time iterative decoding algorithm but this algorithm does not always recover the codeword. Successfulness of iterative decoder depends on the structures in the parity-check matrix called stopping sets.

We tested a method of decoding that adaptively tries to remove stopping sets by using a technique described in [Olmos, Murillo-Fuentes, Perez-Cruz, 2012]. First results seem discouraging for this method but further analysis is needed to say anything conclusive.

We proposed a method called three elimination. For longer codes this decoding algorithm too did not seem to decrease the bit error rate by a lot.

References

[Gallager, 1962] Gallager, R. G. (1962), *Low-density parity-check codes*, IRE Transactions on Information Theory, vol. IT-8, pp. 21-28.

[Cevrero, Leblebici, Ienne, Burg, 2010] Cevrero, A. Leblebici, Y., Ienne, P. Burg, A. (2010), *A 5.35 mm2 10GBASE-T Ethernet LDPC Decoder Chip in 90 nm CMOS*, IEEE Asian Solid-State Circuits Conference.

[Hardesty, 2010] Hardesty, L. (2010), *Explained: Gallager codes.* Available from: http://newsoffice.mit.edu/2010/gallager-codes-0121. [2. may 2015]

[Burshtein, Miller, 2004] Burshtein, D. Miller, G. (2004), *An Efficient Maximum Likelihood Decoding of LDPC Codes Over the Binary Erasure Channel*, IEEE Transactions On Information Theory, Vol 50, No 11.

[Olmos, Murillo-Fuentes, Perez-Cruz, 2012] Olmos, P. M. Murillo-Fuentes, J. J. Perez-Cruz, F. (2012), *Tree-Structure Expectation Propagation for LDPC Decoding over the BEC*, IEEE Transactions On Information theory, Vol 59, No 6.