EE5161:Modern Coding Theory Project

Gaurav Singh, *EE14B125* Nived Rajaraman, *EE14B040*

***Abstract*—In the project we look into designing irregular *LDPC* codes for *BEC* channels using a linear optimization technique based on EXIT charts. We then present the theoretical performance of this code and how it fares against the regular $(3, 6)$ ensemble of *LDPC* codes. Following this, we construct the LDPC decoder and observe the performance of the generated code over different code sizes and versus the *BEC* channel parameter. This report of the project introduces *LDPC* codes and their design, construction, performance and decoding. Each subsection is followed by a detailed unit describing the approach to various problems observed in practice and the results observed from the implementation of the code.**

I. Introduction

Low-density parity-check (LDPC) codes, first proposed by Gallager in his 1962 PhD thesis are a class of high performance, low implementation complexity codes. LDPC codes are usually long codes and are often constructed pseudo-randomly, with only a probabilistic notion of their expected error correction performance. Theoretically, LDPC codes are capacity achieving for the BEC channel and perform very well in practice for long code lengths.

The decoder that is most often paired with LDPC codes is a variant of the sum-product decoder which is an iterative, message-passing based algorithm that is based on soft belief propagation. It is important to note that, when considering the properties of an LDPC code, we are actually considering the properties of a particular choice of parity-check matrix for that code; a different choice of parity-check matrix for the same code might behave differently with respect to sumproduct decoding. This is in contrast with ML decoding, where the probability of incorrect decoding is determined solely by the weight distribution of the codewords.

LDPC codes are either regular (ω_c, ω_r), with bit and check nodes all having the same degree ω_c and ω_r respectively, or are irregular with a distribution for the degrees of the bit and check nodes. They can alternatively be represented by the edge distributions (fraction of edges connected to nodes of given degrees) from the bit-node ($\lambda(x)$) and check-node ($\rho(x)$) perspective. Fixing $\lambda(x)$ and $\rho(x)$ does not fix the code, it generates an ensemble of codes with different lengths. The rate of a regular code can be given by:

$$R_{reg} = 1 - \frac{\omega_c}{\omega_r}, \qquad R_{irr} = 1 - \frac{\sum_{i=2} \lambda_i / i}{\sum_{i=2} \rho_i / i}$$

A. Designing the LDPC codes using EXIT charts

EXIT (**Ex**trinsic **I**nformation **T**ransfer) charts, originally developed in the Turbo decoding community can be adapted for analysis of the theoretical performance of LDPC codes. Similar to Turbo codes with component codes exchanging extrinsic information between each other about the transmitted codeword, LDPC codes can be modelled as single parity check decoders exchanging extrinsic information with a repetition code decoder. We plot the mutual information between the transmitted codeword and the first type of component decoder and the second type of component decoder as the EXIT chart. For the BEC channel, the area between the curves is directly related to the rate of the code. For the optimal decoder, both the curves would coincide and for any feasible error rate, the two curves would form a channel between them where decoding is done iteratively.

The result of EXIT charts is used alongside a linear programming based algorithm to approach the best possible edge distributions from the bit-node (λ) and from the check-node (ρ) perspective. We assume that one of the two is known (in this case, ρ) and compute the best λ and converging to the required rate. An algorithm for the same is proposed as below:

Algorithm 1 Linear programming algorithm to design λ

Input: R (target rate), ρ, TOL ($10\times$ maximum error in rate)
Output: λ, ϵ

Initialization : $\epsilon = 0.1, \Delta = 1000$
while $\Delta >$ TOL **do**
  $\lambda(x)$ = argmax $\sum_{i=2} \frac{\lambda_i}{i}$ subject to:
    i. $1 - \epsilon.\lambda(1-x) < \rho^{-1}(x) \qquad \forall\ x \in [0, 1]$
    ii. $\sum_{i=2} \lambda_i = 1$
  R = $\dfrac{\sum_i \rho_i / i}{\sum_i \lambda_i / i}$
  $\Delta = 0.1(R - 0.5)$
  $\epsilon = \epsilon + \Delta$
end while
return λ, ϵ

The maximization step in the above problem can be represented as a linear programming problem with a small modification. The function constraint on λ, given by: $1 - \epsilon.\lambda(1-x) < \rho^{-1}(x) \quad \forall\ x \in [0, 1]$ can be written as a number of linear constraints once we sample x to lie on a very fine grid between $[0, 1]$. Thus we can use any standard linear programming technique to solve for the optimal $\lambda(x)$ for a given $\rho(x)$ and rate. In order to optimize for both unknown $\rho(x)$ and $\lambda(x)$, the technique of alternating direction minimization can be used to converge to the best distributions (with a limit on the maximum degree). In our case, since $\rho(x)$ was constrained to have 1 or at most 2 types of check node degrees, the optimal $\rho(x)$ was computed by manual iteration. On running this algorithm for a sufficiently small tolerance (TOL), an accurate representation of $\lambda(x)$ can be achieved for the given rate. Figures 1 and 2 illustrate the EXIT charts of the computed optimal $\rho(x)$ and $\lambda(x)$ for different values of ϵ.

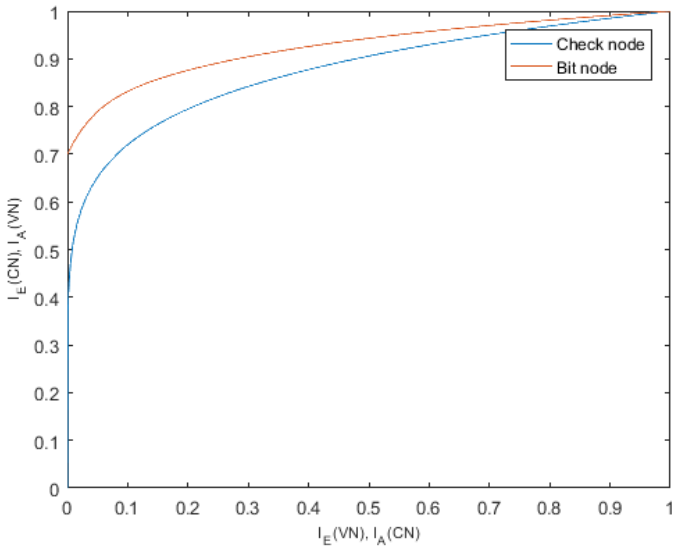

Fig. 1: EXIT chart for $BEC(0.3)$

The above figure illustrates the EXIT chart for the code generated for a channel parameter, $\epsilon = 0.3$. Note that the curve for the bit node does not start from 0 as the bit node has extrinsic information coming from the channel as well. The area in between both curves is where the iterative decoding takes place. The figure below illustrates the same for $\epsilon = 0.495$. As this is the maximum tolerable ϵ, the two graphs are coincident and the area in between converges to 0.

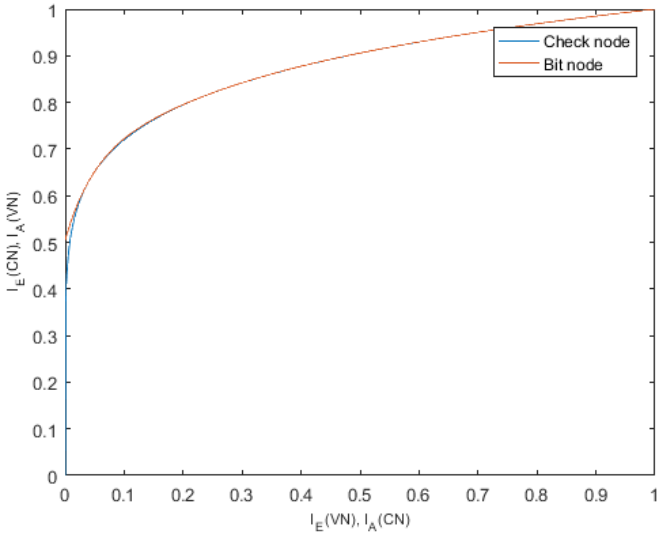

Fig. 2: EXIT chart for $BEC(0.495)$

The experimentally computed $\lambda(x)$ and $\rho(x)$ are seen to be:

$$\lambda(x) = 0.28x^{24}+0.0026x^{7}+0.207x^{6}+0.085x^{3}+0.13x^{2}+0.29x$$

$$\rho(x) = x^{7}$$

B. Density Evolution of LDPC code and its comparison with a regular LDPC code

Over a memory-less channel, under the assumption that the Tanner graphs are all cycle-free (tree structure), it is possible to determine the expected behavior of the iterative sum-product decoding algorithm (or its variants) over a given ensemble. The cycle-free assumption may not be reasonable as the parity check matrix increases in density, but for large code lengths to a degree, can be assumed to be true. This analysis is achieved using a technique known as density evolution. Density evolution tabulates the evolution of the error probabilities in the decoder, taking the expectation not only over all members of the ensemble but also over all possible realizations of the channel noise. To simplify the computation of density evolution into a closed form solution requires a few of constraints on the iterative ensembles. These are as follows:

1) Symmetric Channel: all-zero codeword can be assumed
2) Cycle-free graphs

For binary erasure channel(BEC) a transmitted bit is either correctly received or completely erased, so the density to be tracked is simply the erasure probability. Thus for the BEC the probability density function is effectively one-dimensional, resulting in a greatly simplified analysis. Figure 3 illustrates the density evolution for the optimal code for different values of ϵ. The **theoretically observed error threshold is seen to be $\epsilon = \mathbf{0.4952}$** which is significantly higher than the computed error threshold for the $(3, 6)$ regular ensemble, which is seen to be $\epsilon = 0.4365$.

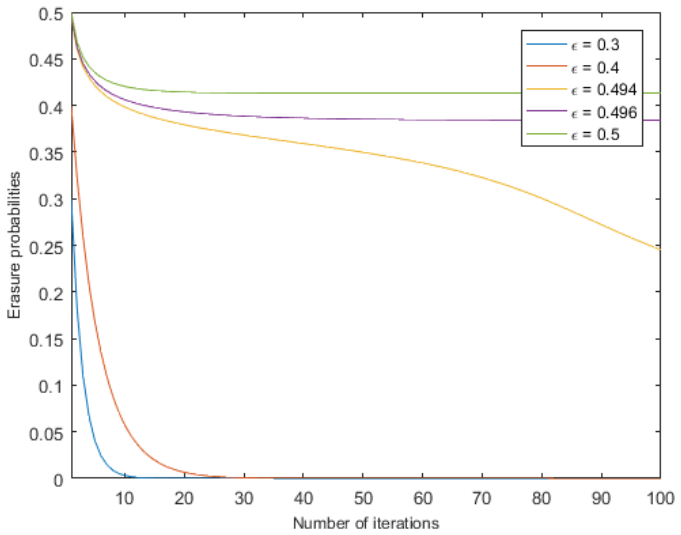

Fig. 3: Density Evolution for different values of ϵ

C. Construction of parity check matrix from the derived edge distributions

As mentioned previously, the edge distributions define the code, but the performance of the code is dependent on the construction of the parity check matrix which could make or break the performance of the code. The construction of the

parity check matrix is done using a very simple variation of the Neal-Mackay method to avoid 4-cycles. The algorithm is as follows:

Algorithm 2 Generating LDPC matrix from P and Λ

Input: Λ, P, n, k
Output: LDPCmat
Initialization : LDPCmat = zeros(n,1)
for col = 1 to n **do**
  badSet = queryRow(LDPCmat, rho)
  for i = 1 to weight(col) **do**
    row = sample from $1 : n-k \sim$ badSet
    LDPCmat(col,row) = 1
    badSet $\cup =$ query4C(LDPCmat, row)
  end for
end for
return Λ, ϵ

Here, the function 'queryRow' outputs the rows that are saturated. In other words, it lists the check nodes that are already connected to the required number of edges. Here, $\cup =$ denotes appending to the LHS variable via set union. The function 'weight' outputs the number of edges to be connected to a particular bit-node, in other words, the weight of the column being filled. Both these functions are trivial to implement. Below we detail the function 'query4C' that lists the locations in a column which should be avoided due to creation of 4-cycles.

Algorithm 3 query4C

Input: LDPCmat (partially completed LDPC matrix), row
Output: badSet
Initialization : badSet = [], potBadCols = []
for i = 1 to n **do**
  if LDPCmat(row, i) = 1 **then**
    potBadCols $\cup = i$
  end if
end for
for j = 1 to $n-k$ **do**
  if sum(LDPCM(j, potBadCols)) $\neq 0$ **then**
    badSet $\cup =$ j
  end if
end for
badSet $\cup =$ row
return badSet

In practice certain modifications have been made to the algorithm ensure that it always converges even if it requires slightly modifying the degree distributions. This is required, since when we compute the degree distributions from the edge distributions, we must round off the former in such a way that the sums are equal to n and $n-k$ for bit and check node degree distributions respectively. However, rounding off in such a way could potentially lead to the count of edges from bit and check node sides being off by a few. Thus, the degree distributions are then re-calibrated in such a way that the edge counts are also equal from either side. This might equate to removing higher degree nodes and replacing them by lower degree nodes, or vice versa. Yer another issue occurring in practice is if for a particular column (more likely for higher weight columns), there is no place to add a new edge simply because the set of saturated rows, plus the set of rows where a 4 cycle could occur put together exhaust the entire row-index space. To tackle this problem, the column in question is eliminated and refilled. Since the filling of edges is at random, there is a possibility of a valid column-index set being found on the 2^{nd} (or higher) attempt. This restart technique is done a set number of times. If failure is reported after all the iterations, the $P(x)$ polynomial is modified and an edge is allowed to be added at a row where no 4-cycles would exist. The modification is always done by modifying a lower degree check node to allow the addition of 1 new edge to it. This procedure would ensure that the column weights are always matched (bit-nodes degrees are satisfied), while the row weights would deviate slightly. This is recommended over the alternate method of modifying the bit-node degree distribution and keeping the check-node distribution intact, since it was observed that more than half of the bit nodes were of degree 2 and making a sufficient number of them to degree 1 could potentially cause those particular bit nodes to no longer 'solve' stopping sets in the Tanner graph.

D. Simulating the constructed codes over BEC(ϵ) using sum-product algorithm

The sum-product algorithm is an iterative algorithm that estimates the transmitted codeword using a message-passing structure. Due to the BEC being a symmetric and memory-less channel, we can assume that the all-zero codeword is transmitted. Based on the channel pararmeter, ϵ, random bits are erased and transmitted. This received vector is passed through the LDPC decoder. Then, the recieved vector is decoder using the following algorithm:

1) **Initialization:** Send $\mu_{i\to j} = y_i$, $\forall i$ and $\forall j \in N_i$
2) **Check Node Update:** $\forall j$ and $\forall i \in N_j$

$$\mu_{i\to j} = \begin{cases} e & \text{if } \exists k \in N_j \backslash i \quad \ni \mu_{k\to j} = e \\ \underset{k\in N_j\backslash i}{\oplus} \mu_{k\to j} \end{cases} \tag{1}$$

3) **Bit Node Update:** $\forall i$ and $\forall j \in N_i$

$$\mu_{i\to j} = \begin{cases} 0 & \text{if } \exists k \in N_i \backslash j \quad \ni \mu_{k\to i} = 0 \\ 1 & \text{if } \exists k \in N_i \backslash j \quad \ni \mu_{k\to i} = 1 \\ e & \textit{otherwise} \end{cases} \tag{2}$$

4) **Termination:** Repeat steps 2 and 3 until all erasures are removed or up-to a maximum number of iterations.

The decoder is constructed from the LDPC matrix generated in the previous step. Edges are connected between bit and check nodes depending on the locations of the 1's in the matrix. The sum product algorithm is run till a maximum number of iterations and is terminated early if the erasures are all found to be removed. If the number of iterations is large enough, and the number of erasures are large enough, the algorithm could potentially fall into a stopping set. If the

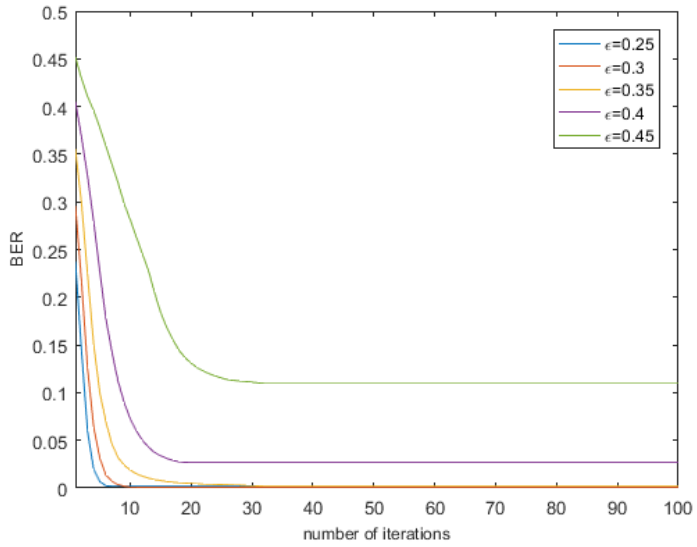

Fig. 4: BER vs. ϵ as number of iterations is increased

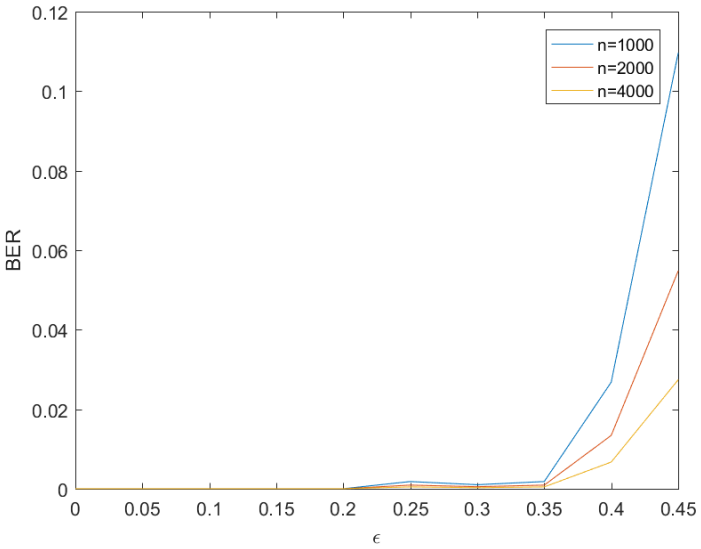

Fig. 5: BER vs. ϵ for different code lengths

number and locations of erasures are favourable, the decoder converges to the correct codeword, eliminating all the erasures. The performance of the decoder for different code lengths is shown below. Note that this is not a very accurate description of BER's as the averaging over noise was only performed for 5 received vectors, due to the long execution time.

II. Conclusion

Given the rate, we are aware that the theoretical limit(Shannon limit) for the bit error rate is 0.49, however, after the implementation of the LDPC code in BEC, we have found out the limit of the bit error rate to be around 0.35. Even though is not equal to the theoretical limit due to practical constraints on the code length and presence of cycles in the Tanner graph, it is still good enough to make the code robust. Improvements to the approach for designing such codes can be done by parallel implementation of the decoder where message passing for different nodes happens on parallel threads of a CPU leading to a huge improvement in the computational complexity.

References

[1] Sarah. J. Johnson, *Iterative Error Correction*, 1st ed. Cambridge University Press, 2009.

[2] J. Campello, D.S. Modha, S. Rajagopalan, *Designing LDPC codes using bit-filling*, Communications, 2001. ICC 2001.

[3] Jun Fan, Yang Xiao, Kiseon Kim, *Designing LDPC codes without cycles of length* 4 *and* 6, Research Letters in Communications, Volume 2008