IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735. Volume 6, Issue 6 (Jul. - Aug. 2013), PP 11-18
www.iosrjournals.org

Comparison of Turbo Codes and Low Density Parity Check Codes

Ahmad Hasan Khan[1], Dr K C Roy[2],
[1]*Electronics and Communication Engg. Suresh Gyan Vihar University, Jaipur*
[2]*Principal SBCET Jaipur*

Abstract-*The most powerful channel coding schemes, namely, those based on turbo codes and LPDC (Low density parity check) codes have in common principle of iterative decoding. Shannon's predictions for optimal codes would imply random like codes, intuitively implying that the decoding operation on these codes would be prohibitively complex. A brief comparison of Turbo codes and LDPC codes will be given in this section, both in term of performance and complexity. In order to give a fair comparison of the codes, we use codes of the same input word length when comparing. The rate of both codes is R = 1/2. However, the Berrou's coding scheme could be constructed by combining two or more simple codes. These codes could then be decoded separately, whilst exchanging probabilistic, or uncertainty, information about the quality of the decoding of each bit to each other. This implied that complex codes had now become practical. This discovery triggered a series of new, focused research programmes, and prominent researchers devoted their time to this new area.. Leading on from the work from Turbo codes, MacKay at the University of Cambridge revisited some 35 year old work originally undertaken by Gallagher [5], who had constructed a class of codes dubbed Low Density Parity Check (LDPC) codes. Building on the increased understanding on iterative decoding and probability propagation on graphs that led on from the work on Turbo codes, MacKay could now show that Low Density Parity Check (LDPC) codes could be decoded in a similar manner to Turbo codes, and may actually be able to beat the Turbo codes [6]. As a review, this paper will consider both these classes of codes, and compare the performance and the complexity of these codes. A description of both classes of codes will be given.*

I. Turbo Codes

There are different ways of constructing Turbo codes, but the commonality is that they use two or more codes in the encoding process. The constituent encoders are typically recursive, systematic convolutional (RSC) codes [3]. Figure 1 shows a general set-up of the encoder. The number of bits in the code word varies with the rate of the code. For an overall code rate of R = 1/2, it is normal to use two R = 1/2 RSC codes, where every systematic bit is transmitted, but for the coded bits a puncturing matrix is used where only every second code bit from each of the constituent codes is included in the transmitted symbol. The permutation matrix Π permutes the systematic bits, so that the two constituent codes are excited by the same set of information bits, albeit in a different order. The permutation matrix is a critical design factor for designing good codes. The code word consists of C_i^1 as the systematic bit (di) and C_i^2 and C_i^3 which are vectors of encoded bits from the two convolution codes. These vectors can contain one or more bits and di is the information bit, Π is the permutation matrix.

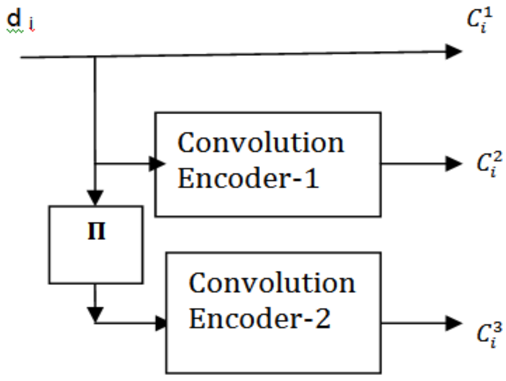

Figure 1: Block Diagram of Turbo Encoder

1.1. The BCJR algorithm

The BCJR algorithm [1] is used for optimal decoding of the constituent codes of the turbo codes. This algorithm is similar to the message passing algorithm used in the context of LDPC codes. The BCJR algorithm is a symbol based decoder, and delivers probabilities of the correctness of the symbol decoding together with each decoded bit. In a convolution code, we must separate between state transitions caused by an information bit having value 1 and those caused by an information bit having value 0. We denote the states linked together by a state transition that occurs between time (i – 1) and time i S_{i-1}^{a1} and S_i^{b1} if this transition is caused by a 1 as the information bit. Correspondingly, the notation is S_{i-1}^{a0} and S_i^{b0} if the transition is caused by a 0 as the information bit. We denote the vector of received channel symbols y, where y is perturbed by AWGN. We denote the information bit at time i di. The log likelihood ratio for a state transition associated with di = 1 and di = 0 is then given by

$$\lambda_i = \log\frac{P(S_{i-1}^{a1}, S_i^{b1}, y)}{P(S_{i-1}^{a0}, S_i^{b0}, y)} \qquad (1)$$

In the case of many possible state transitions, the expression gets more complicated:

$$\lambda_i = \log\frac{\sum_{S_{i-1}^{a}, S_{i|di=1}^{b}} P(S_{i-1}^{a}, S_i^{b}, y)}{\sum_{S_{i-1}^{a}, S_i^{b}|di=0} P(S_{i-1}^{a}, S_i^{b}, y)} \qquad (2)$$

The vector y may be split into three parts:

$$y = \sum_{p=1}^{i=1} y_p + y_i + \sum_{k=i+1}^{N} y_k \qquad (3)$$

s_i^b And y_i depend only on the state s_{i-1}^a and the bit d_i, whereas $\sum_{k=i+1}^{N} y_k$ depend only on s_i^b and the bits d_k k $\epsilon\{i+1, N\}$. The probability of the state transition s_{i-1}^a s_i^b is then given by the expression in (4).

$$P(s_{i-1}^a, s_i^b y,)$$
$$= P(s_{i-1}^a, s_i^b, \sum_{p=1}^{i=1} \mathbf{y_p} + y_i + \sum_{k=i+1}^{N} y_k \qquad (4)$$

$$= P(s_{i-1}^a, \sum_{p=1}^{i=1} y_p)(y_i i + \sum_{k=i+1}^{N} y_k, S_i^b | S_{i-1}^a)$$
$$= P(s_{i-1}^a, \sum_{p=1}^{i-1} y_p)(y_i, S_i^b | S_{i-1}^a)(\sum_{k=i+1}^{N} y_k | S_i^b)$$

Eqn. 4 may then be written as

$$P(s_{i-1}^a, s_i^b y) = \alpha(s_{i-1}^a)\gamma(s_{i-1}^a, s_i^b)\beta(S_i^b) \qquad (5)$$

With the following definitions:

$$\gamma(s_{i-1}^a, s_i^b) \triangleq P(y_i, s_{i-1}^a | s_i^b,)$$
$$= P(y_i | s_{i-1}^a, s_i^b,)\, P(s_{i-1}^a, s_i^b,) \qquad (6)$$
$$\alpha(s_{i-1}^a) \triangleq P(s_{i-1}^a, \sum_{p=1}^{i-1} y_p) \qquad (7)$$

$$\beta(S_i^b) \triangleq (\sum_{k=i+1}^{N} y_k | S_i^b \qquad (8)$$

Furthermore, we may express

$$\alpha(s_{i-1}^a) = \sum_{s_{i-2}^a} \alpha(s_{i-2}^a)\, \gamma(s_{i-2}^a, s_{i-1}^b) \qquad (9)$$

And

$$\beta(S_i^b) = \sum_{s_{i-1}^a} \beta(s_{i+1}^a)\, \gamma(s_i^a s_{i+1}^b) \quad (\qquad 10)$$

We substitute Eqn. (5) into Eqn. (2) and obtain Eqn. (11).

$$\lambda_i = \log\frac{\sum_{S_{i-1}^{a}, S_{i|di=1}^{b}} \alpha(s_{i-1}^a)\gamma(s_{i-1}^a, s_i^b)\beta(S_i^b)}{\sum_{S_{i-1}^{a}, S_i^{b}|di=0} \alpha(s_{i-1}^a)\gamma(s_{i-1}^a, s_i^b)\beta(S_i^b)} \qquad (11)$$

As shown in Eqn. (6), we have:

$$\gamma(s_{i-1}^a, s_i^b) = P(s_{i-1}^b | s_i^a,)\, P(y_i | s_i^b, s_{i-1}^a,) \qquad (12)$$

Through knowledge of the nature of AWGN, Eqn. (12) may be written as:

$$\gamma(s_{i-1}^a, s_i^b) = P(s_{i-1}^b | s_i^a) \prod_{j=1}^{n} e^{-\frac{((y_i^j)(2c_i^j-1)^2)}{2\sigma^2}} \qquad (13)$$

We compute the logarithm of Eqn. (13) and obtain Eqn. (14).

$\log\gamma(s_{i-1}^a, s_i^b) = P(\log(s_{i-1}^b | s_i^a)) + \sum_{j=1}^{n} \frac{(y_i^j - a_i^j)^2}{2\sigma^2} - \log(\sqrt{(2\,\sigma^2)})$ (14)

P $(s_{i-1}^b | s_i^a)$ is equal for all permissible states Sb if we assume an equal probability of 1's and 0's in the information sequence. This part may therefore be omitted in the decoding process. The metric of a symbol n is then given by Eqn. (43)(15).

$M_i^{ab} = \sum_{j=1}^{n} \frac{(y_i^j)^2 + (a_i^j)^2 + 2y_i^j a_i^j}{2\sigma^2}$- (15)

1.2. The Log-BCJR algorithm

The Log-BCJR algorithm is a simplification of the BCJR algorithm in terms of the needed computations. We start with the following definitions:

$f(\cdot) \triangleq \log\alpha(\cdot)$

$r(\cdot) \triangleq \log\beta(\cdot)$

$g(\cdot) \triangleq \log\gamma(\cdot)$

$f(s_i^b) = \log(\sum_{s_{i-1,}^a} \alpha(s_{i-1,}^a)\ \gamma(s_{i-1,}^a s_i^b)$ (16)

$\log(\alpha(s_{i-1,}^a) = f(\alpha(s_{i-1,}^a)) \Rightarrow \alpha(s_{i-1,}^a))$

$= e^{f(s_{i-1,}^a)}$ (17)

This gives:

$f(s_{i,}^a) = \log(\sum_{s_{i-1,}^a} e^{f(r(s_{i-1,}^a)+g(s_{i-1,}^a, s_i^b}))$ (18)

Correspondingly for β :

$r(s_{i-1)}^b = \log(\sum_{s_{i,}^b} e^{(r(s_{i,}^b)+g(s_{i-1,}^a, s_i^b}))$ (19)

Eqn. (20) shows the expression for $\log\gamma(s_{i-1}^a, s_i^b)$.

$\gamma(s_{i-1,}^a s_i^b) = \log(P(y_i | (s_i^b s_{i-1,}^a) + \log(P(s_i^b | s_{i-1}^a))$ (20)

γ Now becomes:

$\gamma i = \log((\sum_{s_{i-1,,}^a s_i^b |di=1} e^{(f(s_{i,-1}^a)+r(s_i^b)+g(s_{i-1,}^a, s_i^b}))$

$- \log((\sum_{s_{i-1,,}^a s_i^b |di=0} e^{(f(s_{i,-1}^a)+r(s_i^b)+g(s_{i-1,}^a, s_i^b}))$

(21)

This implies a summation of a series of exponentials. This may be simplified(4) :

$\mathrm{Log}(e^x + e^y) = \max(x, y) + \log(1 + e^{|x-y|})$ (22)

We denote this calculation L(x, y).

$\log\sum_{i=1}^{I} e^{xi} = \log(e^{xi} + \log\sum_{i=2}^{I} e^{xi}$

$= \log(e^{xi} + \log\sum_{i=1}^{I-1} e^{xi}$

$= L(xi, \log\sum_{i=1}^{I-1} e^{xi})$

$= L(xi, \log(e^{I-1} + \sum_{i=1}^{I-2} e^{xi})$

$= L(xi, L(xI-1, \log(\sum_{i=1}^{I-2} e^{xi}))$ (23)

This becomes a recursion which we may write as Eqn. (24).

$\log\sum_{i=1}^{I} e^{xi} = L(xI, L(xI-1, L(\ldots L(x1, x1)\ldots)))$ (24)

In most cases, this may be further simplified as shown in Eqn. (25).

$\mathrm{Log}\sum_{i=1}^{I} e^{xi} \approx \max(xi)$ (25)

Through the use of this simplification, the computation of f(·) and r(·) becomes a forward and backward Viterbi algorithm respectively

.

1.3. Decoding of Turbo Codes

The decoding algorithms described in the preceding sections are algorithms for decoding the constituent RSC codes of turbo codes. In order to get the 'turbo structure' the decoding algorithms of each of the constituent codes have to be combined in a similar manner to the message passing algorithm described previously. Figure 2 illustrates a multiple iteration decoder. Here, there is a feedback loop, where the first decoder is fed new information from the last decoder of the previous iteration. The information that is passed on to the next decoder is generally denoted γ e. This is the extrinsic part of γ that was generated in the given decoding. The subtractions shown in the figure ensure that only new information is used in subsequent decoding stages. This message passing implies that the a posteriori probability from one decoder is used as a priori information in the subsequent decoder.

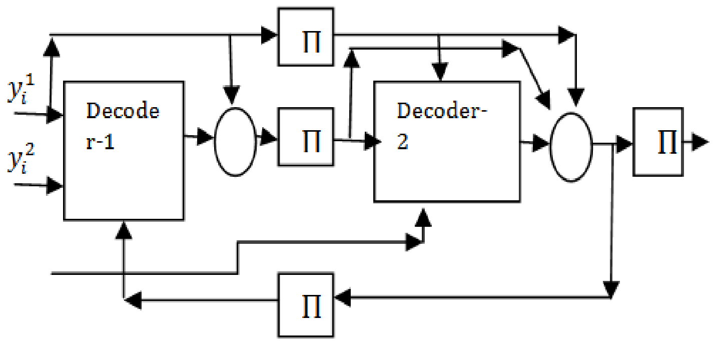

Figure (2) Block Diagram of Turbo Decoder, several iteration

II. Low Density Parity Check Codes

The structure of the code is given by a parity check matrix H, illustrated below. Different codes have different parity check matrices'.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad (26)$$

The code word x is constructed so that Eqn. (27) is true.

$$Hx = 0 \qquad (\text{mod } 2) \qquad (27)$$

A generator matrix G is used for constructing the code. The generator matrix may be found from the parity check matrix H.

2.1. Construction of G

First we note that

$$H_X = \quad X^T H^T \qquad (28)$$

The code word x may be split into one information part **i** and one parity check part c. The code word may then be written as

$$\mathbf{X}^T = [\mathbf{i}\,]\,[\mathbf{c}] \qquad (29)$$

Correspondingly, the parity check matrix may be split into two matrices:

$$H = [A\,|B] \qquad (30)$$

From Eqn.(28), we note that vector **i** is multiplied with Matrix A, whereas vector c is multiplied with matrix B.
On the basis of Eqn. 29 and30, Eqn.27 may be written as

$$A\,\mathbf{i} + B\,\mathbf{c} = 0 \qquad (31)$$

If the matrix B is non-singular, Eqn. 31 may be inverted and the check bits c may be found from Eqn. (32)

$$\mathbf{c} = \mathbf{B}^{-1}\mathbf{A}\,\mathbf{i} \qquad (33)$$

In practice, it may be necessary to swap over some of the columns in H in order for B to become non-singular. The product $\mathbf{B}^{-1}\mathbf{A}$ makes out the generator matrix G. This matrix is calculated only once, and is used for all encoding. The parity check matrix is used for constructing a graph structure in the decoder.

2.2. Graph structure

The decoding of LDPC codes may be efficiently performed through the use of a graph structure. In this work, Tanner graphs will be used for the decoding [8]. The graph is constructed from the parity check matrix H.

Each row in the matrix is represented by a check node, whereas each 1 in the row is represented by an edge into a bit node. Each column is represented by a bit node, and each 1 in the column corresponds to an edge into a check node. This is illustrated in Figures 3 and 2. In this manner, a graph is constructed which contains a total of N bit nodes and M check nodes. The number of edges is decided by the number of 1's in the parity check matrix. All edges are connected to a check node and to a bit node. The number of edges connected to a node denotes the degree of the node.

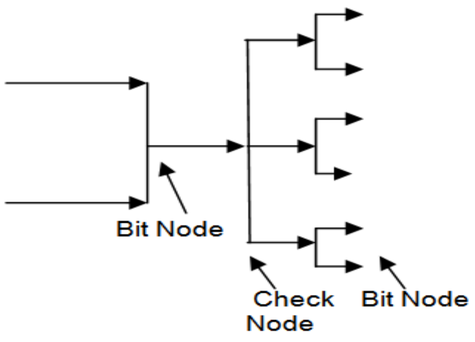

Figure 3: Check Node

2.3. Decoding

In this context, the decoder is a soft-decision input decoder, implying that it operates on the channel symbols, denoted by

$r = 2x - 1 + n$ (34)

Where n is the AWGN noise vector added in the channel and x is the code word.

2.4. The parity of a vector

Finding the probability of the parity of a vector is a central concept in the decoding of LDPC codes. Each parity check may be regarded a vector of even parity [2]. First, we define the Likelihood Ratio (LR) as the ratio between the two probabilities P(x = 1) and P(x = 0):

$$LR = \frac{P(X=1)}{P(X=0)} \quad (35)$$

The symbol γ is used for the Log Likelihood Ratio,

$$\gamma = \log\frac{P(x=1)}{P(x=0)} \quad (36)$$

If x is a vector of bits, the LLR of a bit i in that vector is given by γ i.

$$\gamma_i = \log\frac{P(xi=1)}{P(xi=0)} \quad (37)$$

The notation $\varphi((x)$ is used for the vector parity. The LLR of the parity of the vector x is then given by:

$$\lambda_{\varphi(x)} = \log\frac{P(\phi(x_i)=1)}{P((\phi(x_i)=0)} \quad (38)$$

In order to compute $\lambda_{\varphi(x)}$ Eqn. 39 is used:

$$\tanh(-\frac{\lambda\,\phi(x)}{2}) = \prod_{i=1}^{n}\tanh\left(-\frac{\lambda i}{2}\right) \quad (39)$$

Eqn. 13[39] is solved with respect to $\lambda\phi$:

$$\lambda\phi_{(x)} = -2\tanh^{-1(} = \prod_{i=1}^{n}\tanh\left(-\frac{\lambda i}{2}\right) \quad (40)$$

The a posteriori LLR of a bit n is given by:

$$\lambda_n = \log\frac{P(x_n=1|r)}{P(x_n=0|r)} \quad (41)$$

The vector r may be split into two parts r_n which refers to the systematic part of the code word, and {ri≠n}, which refers to the parity bits of the code word. Eqn. 41 may then be expressed as

$$\lambda n = \log\frac{P(x_n=1|\,r_n,\{ri\ \neq 1\})}{P(x_n=0|\,r_n\{ri\ \neq 1\})} (42)$$

Bayes rue is given by:

$$P(a/b) = \frac{P(b,a)}{P(b)} \quad (43)$$

We use this rule in order to re-express the numerator of Eqn. 42:

$$P(x_n = 1|\,r_n,\{ri\ \neq 1\} = \frac{f(rn,xn=1,[ri\neq 1\})}{f(rn,,[ri\neq 1))}$$

(44)

Furthermore, using the equality P (a, b) = P (a |b)P(b):

$$P(x_n = 1|\, r_n, \{ri \neq 1\,) = \frac{f(rn, xn = 1|, [ri \neq 1\})f(xn = 1, \{ri \neq 1\})}{f(rn.|\,, \{ri \neq 1\})f\{ri \neq 1\}}$$

(45)

Given xn, rn is statistically independent of $\{ ri \neq 1\}$:

$$P(xn = 1|rn, \{ri \neq 1\}) = \frac{f(rn.|xn = 1))f(xn = 1, \{ri \neq 1\})}{f(rn.,|\{ri \neq 1\})}$$

(46)

Following the same line of reasoning, the denominator of Eqn. 16 is expanded. Eqn. 42 then becomes:

$$\lambda_n = \log \frac{f(rn.|xn=1)P(xn=1,|\{ri \neq 1\})}{f(rn.|xn=0)P(xn=0,|\{ri \neq 1\}}$$

$$= \log \frac{f(rn.|xn=1)}{f(rn.|xn=0)} + \log \frac{P(xn=1,|\{ri \neq 1\})}{P(xn=0,|\{ri \neq 1\})} \qquad (47)$$

In an AWGN channel the conditional probability
$f(r_n|x_n)$ is given by

$$f(r_n|\, x_{n)} = \frac{1}{\sqrt{2\pi\sigma}}\, e^{-\frac{(rn-2xn+1)}{2\sigma}} \qquad (48)$$

Substituting Eqn. 22 into Eqn. 21 we get Eqn. 49.

$$\lambda_n = \frac{2}{\sigma 2} rn + \log \frac{P(xn=1,|\{ri \neq 1\})}{P(xn=0,|\{ri \neq 1\})} \qquad (49)$$

The first element of the right hand side of the equal sign is denoted the intrinsic part, whereas the second element is denoted the extrinsic part. Note that if it is known that the parity of a vector x is 0 (even parity), the probability that a bit xn is 1, given the received values of the rest of the vector ri $\neq$ n, is the same as the probability that the rest of the vector $\{ ri \neq n\}$ has odd parity.

$$\lambda_n = \frac{2}{\sigma 2} rn + \log \frac{P(\phi\; xi=1 for\; i=1....j,|\{ri \neq n\})}{P(\phi\; xi=0 for\; i=0....j,|\{ri \neq 1n\})}$$

(50)

If the graph is free of cycles, the vectors x1,x2..xj are

Independent, given$\{ ri \neq 1n \}$, and if the elements within the vector are independent, Eqn. 50 may be written as Eqn. 51.

$$\lambda_n = \frac{2}{\sigma 2} rn + \log \frac{\prod_{i=1}^{j} P(\phi\; xi=1\{ri \neq n\})}{\prod_{i=1}^{j} P(\phi\; xi=0|\{ri \neq n\})}$$

$$= \frac{2}{\sigma 2} rn + \sum_{i=1}^{j} \log \frac{P(\phi\; xi=1\{ri \neq n\}}{P(\phi\; xi=0|\{ri \neq n\})}$$

$$\frac{2}{\sigma 2} rn + \sum_{i=1}^{j} \lambda\; \phi(xi) \qquad (51)$$

In the graph λ i, l is the message (contribution) from bit node i to check node l:

$$\lambda\, i, l = \log \frac{P(\phi\; xi=1\{ri \neq n\}}{P(\phi\; xi=0|\{ri \neq n\})} \qquad (52)$$

We substitute Eqn. 14 into Eqn. 25

$$\lambda_n = \frac{2}{\sigma 2} rn - 2\sum_{i=1}^{j} \tanh^{-1}\left(\prod_{l=2}^{k} \tanh\left(-\frac{\lambda\, i,l}{2}\right)\right) \qquad (53)$$

Eqn. 54 is the expression of the LLR of bit n. The first Part is the intrinsic information, whereas the second part is the extrinsic information from the j vectors that contain bit n.

Messages into bit node n each bit node has a particular LLR associated with itself. The LLR is computed from the contributions from all connected check nodes, which constitute the extrinsic information, and the received values rn, which constitute the intrinsic information. When a check node k receives a message from a bit node i, it subtracts the value it passed on to bit node i in the previous iteration, i.e. $\lambda_{i,l=}\lambda_{i-}\mathbf{u}_{m,i}$. This is done in order to reduce the correlations with previous iterations.

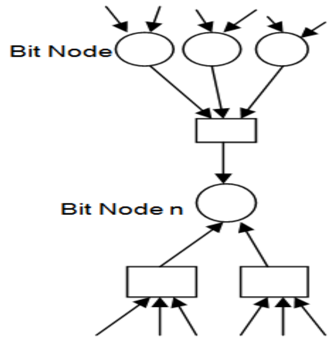

Figure 4: Messages into bit node n

III. Comparison Of Turbo Codes And Low Density Parity Check Codes

A brief comparison of Turbo codes and LDPC codes will be given in this section, both in term of performance and complexity. In order to give a fair comparison of the codes, we use codes of the same input word length when comparing. The rate of both codes is R = 1/2. Figures 5 and 6 show the performance of Turbo codes and the LDPC codes with information length 1784 and 3568 respectively. The Turbo codes defined in 6 were used for obtaining the performance curves. For both code lengths the LDPC codes are better until a certain S/N is reached, respectively Eb/N0 = 1.15 and 1.1 dB. This characteristic 'error floor' or 'knee' was also typical of Turbo codes in earlier years, but this has become less of a problem lately due to the definition of good permutation matrices. In order to compare the complexity of the codes, the number of multiplications, additions and complex operations like log, tanh, e and $\tanh^{-1}$ were counted in both codes. Figure 8 shows the number of operation per iteration for the LDPC and Turbo codes of length I = 1784 and 3568. The number of iterations used may vary in the case of the LDPC codes, as the decoding quality may easily be checked, i.e. that if a valid code word is decoded, the decoding may be stopped. Hence, there will be less and less iterations performed the higher the S/N. Hence, in order to characterize the mcomplexity, a representative number was chosen in order to give a fair comparison with Turbo codes: With the above parameters the bit error rate for the LDPC code was 0.0038 and 0.00054 for the I = 1784 and I = 3568 codes respectively, whereas for the Turbo code the bit error rate was 0.0043 and 0.00062 respectively, i.e. the error rates were sufficiently similar for a fair comparison to be given in Figure 5 and Figure 6.

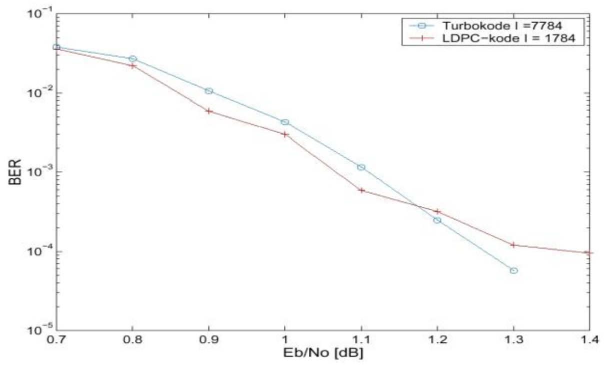

Figure 5: Turbo code and LDPC codes: I = 1784.

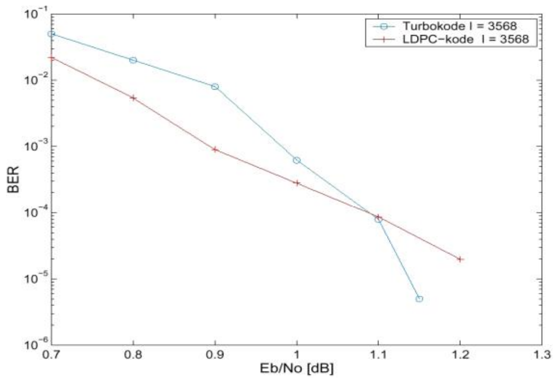

Figure 6: Turbo code and LDPC codes: I = 3568

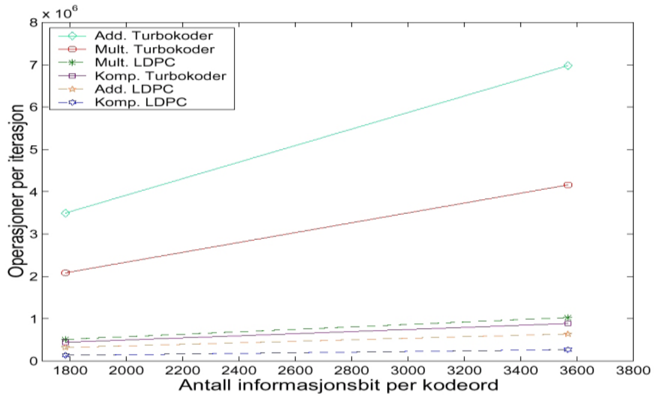

Figure 7: Complexity: The figure shows Additions),Multiplications. and Complex operations.)per iteration for LDPC and Turbo codes for I = 1784 and I = 3568.

IV. Conclusion

The results in the previous section show that LDPC codes have a significantly lower complexity than Turbo codes at the code lengths, performance and S/N that were considered here. Figure 7: Complexity: The figure shows Additions (Add.), Multiplications (Mult.) and Complex operations (Komp.) per iteration for LDPC and Turbo codes for I = 1784 and I = 3568. Turbo codes have a fixed number of iterations in the decoder. This implies that the time spent in the decoding and the bit rate out of the decoder, are constant entities. In contrast, the LDPC decoder stops when a legal code word is found, implying that there is potential for significantly reducing the amount of work to be done relative to Turbo codes. This also implies that the bit rate out of the decoder will vary, and a buffer system must be designed in order to make the bit rate constant. The LDPC decoder will become faster the higher the S/N. An advantage with LDPC codes is that the decoders may be implemented in parallel. This has significant advantages when considering long codes.

References

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, IEEE Transactions on Information Theory, vol. 20, IEEE, 1974, pp. 284–287.
[2] John R. Barry, Low-density parity-check codes, Tech. report, Georgia Institute of Technology, 2001.
[3] C. Berrou, A. Glavieux, and P. Thitimajshima, Near shannon limit error-correcting coding and decoding: Turbo-codes, Proceedings ICC (Geneva), IEEE, May 1993, pp. 1064–1070.
[4] Kjetil Fagervik, Iterative decoding of concatenated codes, Ph.D. thesis, University of Surrey (UNIS), UK, August 1998.
[5] R. G. Gallager, Low density parity check codes, Research monograph series, no. 21, MIT Press, Cambridge, MA, 1963.
[6] David J.C. MacKay and Radford M. Neal, Near Shannon limit performance of low density parity check codes, Electronics Letters 32 (1996), no. 18, 1645– 1646, Reprinted in Electronics Letters, 33 (1997), no. 6, 457-458.