Evaluation of Complexity Versus Performance for Turbo Code and LDPC Under Different Code Rates

Alaa Eldin.S. Hassan
Dept. Space Science
National Authority for Remote sensing and Space Science (NARSS)
Cairo, Egypt.
E-mail: alaa_eldin@narss.sci.eg

Moawad Dessouky, Atef Abou Elazm and Mona Shokair
Dept. Electrical Communication
Menoufiay University
Menouf, Egypt
atef_abuelazm@menofia.edu.eg,
i_shokair@yahoo.com, Dr_moawad@yahoo.com

***Abstract*— Turbo codes play a major role in channel error correction schemes used in wireless communications. Turbo codes emerged in 1993 and since that time they dominate the research in error control coding together with low-density parity-check codes. Due to their remarkable performances, turbo code and low density parity check code have been accepted to a number of standards by many organizations which decide to include turbo code and low density parity check into their new standards after these codes were proven successful in a many of missions. In this paper, the calculation and comparison of performance versus complexity for those two techniques of channel coding was done. For a fair comparison, the performance and complexity should be compared together. The complexity was calculated by counting the number of clock cycles need to complete the decoding algorithm. This comparison is used as a guiding lines of using either turbo code or the low density parity check in specific communication applications, The performance comparison of turbo code and low density parity check were computed for rates 1/2 and 7/8. The complexity for the two codes were calculated for different code rates like (1/2, 1/3, 3/4, 7/8), the evaluation study concludes that the turbo code was recommended for moderate rate, while the LDPC is recommended for higher code rates.**

Keywords-turbo code; LDPC; complexity.

I. INTRODUCTION

Turbo code is a powerful error correcting code used in wireless communication. Turbo code emerged in 1993 [1] and since this year, it becomes a popular area of communications research. Turbo code has a performance near to Shanon limit, and it is stable for long time and now being accepted as standard forward error correction technique by many organizations such as CCSDS, but turbo code still facing high complexity problem, on the other hand Low-density parity-check (LDPC) codes are forward error-correction codes, first proposed in the 1962 by Gallager in his dissertation at MIT [2], at that time it was unpractical to be implemented, but then largely neglected for over 35 years. After that, it is rediscovered again by MacKay and Neal in their work [3]. Because LDPC shares the main concept of message passing algorithm as the turbo code and its performance is also very close to Shannon limit. However, in the last few years, the advances in low-density parity-check codes have prove that the LDPC beat turbo code in terms of error floor and performance for the higher code rates. In this paper, we analyze the decoding algorithm for turbo code and calculating its complexity under different code rates, and then the same scenario was applied for the LDPC to calculate its complexity and performance for different code rates.

A similar work was made at Stravanger University [4], but the comparison was made for rate 1/2 only, in this paper a complete comparison of turbo code and LDPC was made for different code rates. Also, because the complexity of LDPC is a function of code rate, the complexity was studded for different code rates. It is concluded from this research work that the Turbo code is recommended for moderate code rate because of its better performance, while the LDPC is recommended for higher code rates because of its better performance besides lower complexity compared with turbo code.

This paper is organized as follows; the turbo code decoding algorithm is reviewed in Section II and Section III. The complexity calculations are made in Section IV. The LDPC code decoding algorithm is reviewed in Section V. its complexity calculation is made in Section VI. The comparison of performance and complexity was elaborated in Section VII and Section VIII. Tradeoff between performance and complexity was compared in Section IX; finally, conductive conclusions are done in Section X.

II. TURBO CODE SCHEME

Conventional turbo code consists of two (or more) convolutional codes connected in serial or in parallel via some pseudo-random interleavers.

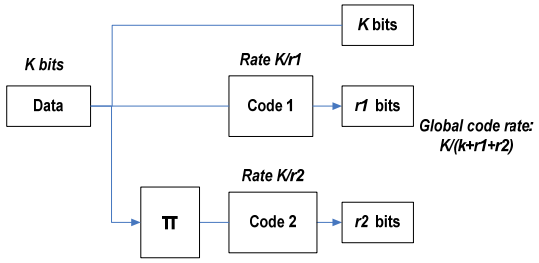

Figure 1. Basic structure of the turbo encoder

 ISBN: 978-1-61208-194-6

Figure 1 presents a block diagram of an encoder of a systematic turbo code with an information block length K. The turbo code is composed of two Recursive Systematic Convolutional codes (RSC) [1].

The information bits are fed to the first RSC and after being interleaved are passed through the second constituent encoder. The resulting codeword consists of the systematic bits, k (i) , and two parity check streams, $r_1(i)$, $r_2(i)$, i = 1, 2, . . . , K . The coding rate of this code is 1/3. Higher code rates can be achieved by puncturing some of the parity check bits, using more constituent codes result in codes with rates lower than 1/3

III. TURBO CODE DECODING ALGORITHM

A turbo decoder consists of two concatenated decoders, each using the received systematic stream and the corresponding received parity stream. Each decoder provides a soft output of the transmitted bits by using the received data and the information provided by the other decoder. The soft output is the a posteriori probability (APP) and consists of three components: the intrinsic information which is a function of the received signal for the corresponding bit position, the a priori (AP) probability of that bit position and the extrinsic information which comes from the received signal for other bit positions and their a priori probabilities, In each iteration the extrinsic information produced by the other constituent decoder is used to evaluate the a priori probabilities in that iteration. Repeating this procedure improves the estimation of the bit probability values and hence, reduces the probability of error. One efficient algorithm for soft output decoding, based on the trellis diagram of the code known as the BCJR algorithm, is presented in [5]. The suboptimal decoder introduced in [6] Finds the extrinsic information on the transmitted bits by one of the constituent decoders and passes it to the other decoder through the interleaver. The decoder can decode the received vector only if the iterative decoding converges. The output of the "symbol-by-symbol" Maximum a posteriori Probability (MAP) decoder is defined as the a posteriori log-likelihood ratio, that is, the logarithm of the ratio of the probabilities of a given bit being "+1"or"-1' given the observation y, as in equation (1). The Max-log MAP algorithm for decoding the turbo code was used as it presented in [7] and it is based on the trellis of a convolutional encoder in Figure 3.

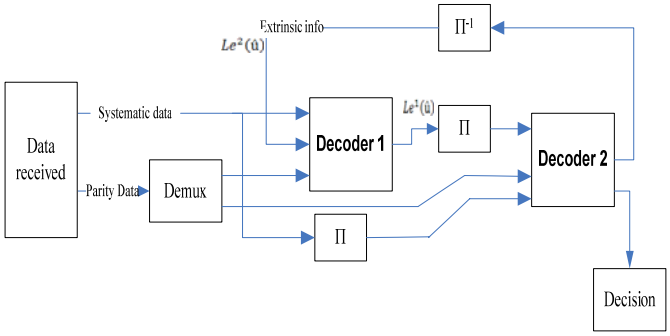

Figure 2. Basic structure of an iterative turbo decoder

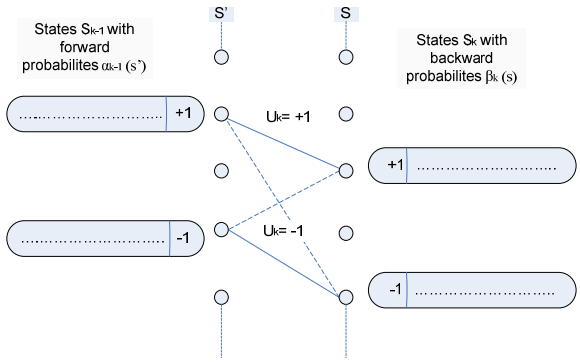

Figure 3. Trellis structure of systematic convolutional codes

The log-likelihood ratio of (u_k) is defined by [7]

$$L(\hat{u}_k) = ln\frac{P(u_k = +1|y)}{P(u_k = -1|y)} = ln\frac{\sum_{u_k=+1}^{(s',s)} P(s',s,y)}{\sum_{u_k=-1}^{(s',s)} P(s',s,y)} \quad (1)$$

$$L(\hat{u}_k) = ln\frac{\sum_{u_k=+1}^{(s',s)} \alpha^*_{k-1}(s').\gamma^*_k(s',s).\beta^*_k(s)}{\sum_{u_k=-1}^{(s',s)} \alpha^*_{k-1}(s').\gamma^*_k(s',s).\beta^*_k(s)} \quad (2)$$

where u_k is the information bit at time k, and α^*_k: is the Probability of path in the trellis going from state S' (k-1) and terminating at state S (k)

$$\alpha^*_k(s) = \sum_{(s,s')} \gamma^*_k(s',s)\,\alpha^*_{k-1}(s') \quad (3)$$

and β^*_k is the Probability of path in the trellis going from state S(k-1) and terminating at state S'(k)

$$\beta^*_{k-1}(s') = \sum_{(s,s')} \gamma^*_k(s',s)\,\beta^*_k(s) \quad (4)$$

and γ^*_k is the branching Probability of path in the trellis going from state S (k-1) and terminating at state S'(k)

$$\gamma^*_k(s',s) = \frac{1}{2}.x_k.L_a.(x_k) \quad + x_k.L_c.x'_k \quad + \quad p_k.L_c.p'_k \quad (5)$$

IV. COMPLIXITY OF TURBO DECODING

In this section, we have to have a specific formula for the complexity needed for Turbo code implementation by counting the number of processor cycles for mathematical operations needed for decoding, which related to the time needed to decode a frame of information encoded by Turbo code. From Equations (3), (4), and (5) for the Max-Log-MAP algorithm, the α^*, β^*, and γ^* have to be calculated. The calculation of α^*_k (u) requires two multiplications and one addition for each state. Assuming the encoder with memory length M, so we have to multiply by the number of states 2^M so, ($2*2^M$) multiplications is needed and 2^M addition. The same number of calculation is needed for β^*_k (u). For the γ^*_k we need ($3*2^M$) multiplications and ($2*2^M$) additions plus ($1*2^M$) divisions.

 ISBN: 978-1-61208-194-6

Let us take turbo code with M = 4 so, 2^M = 16. The decoder has two operations per iteration, so, all number should be multiplied by two.
The total number of additions is (4 * 2 * 16) = 128;
Total number of multiplications is (7 * 2 * 16) = 224;
Total number of divisions for branch metric is (2*16) = 32;

Also, here we assume that max* (a,b) ≈ max (a,b) because the correction factor is small and will reduce the complexity, but its effect on the performance is less than 0.2 dB [7].

So, Max* operation need two cycle per state per iteration.

The total operations per bit is ($2*2*2^M$) = 64;

For the logarithm operation, a look up table was used which need one operation per state per iteration.

The total operations per bit is ($1*2*2^M$) = 32

Finally, the divisions needed to calculate final L (u_k) is per bit.

Here, it is assumed that 3 clock cycles needed for an addition, 5 clock cycles needed for multiplication operation, and 17 clock cycles for a division operation as in the case for a typical Pentium processor [8]. The ratio of calculation complexity is assumed to be as follows: Addition: multiplication: division= 1:1.5:5.

$$224 + 224 * 1.5 + 33 * 5 = 725 \text{ cycles/iteration/ frame length} \quad (6)$$

Table I summarizes the processor cycles for decoding of Turbo code for different code rates, the number of iterations per frame length "K" is 10; as we mention before, the complexity of turbo code is the same as the mother code rate, e.g., rates 1/2 and 3/4 can be obtained from the mother code 1/3, so all of them has the same complexity as the mother code 1/3.

TABLE I. COMPLEXITY OF TURBO CODE FOR DIFFERENT CODE RATES

Code Rate	Processor cycles/ iteration/Frame length	For 10 iterations /Frame length
Rate 1/3	425/ Frame length	7250
Rate 1/2	725/ Frame length	7250
Rate 3/4	725/ Frame length	7250

V. LDPC DECODING PRINCIPLES

Decoding of LDPC used message passing algorithms, these algorithms interpreted by bipartite graph representation of the LDPC code [7], where variable nodes and check nodes are connected through edges. The variable nodes and check nodes exchange the messages along their edges in an iterative fashion, thereby cooperating with each other in the decoding process.

The operations in an LDPC decoder comprises of two steps; first, is how the messages are generated at the check nodes and variable nodes (called "node processing"); second, which determine how the generated messages are passed between each other (called "scheduling"). These two operations determine the decoding complexity of LDPC.

A. Node processing

Node processing consists of variable node update (VNU) and check node update (CNU). In the VNU, incoming messages from the check nodes are processed at each VN, and the outgoing messages are generated and passed to the check nodes. Similarly, in the CNU, incoming messages from the variable nodes are processed at each CN, and the outgoing messages are generated and fed back to the variable nodes. Thus, the messages are passed between the variable nodes and check nodes iteratively.

Let C(n) denote the set of check nodes connected to variable node n, and V(m) denote the set of variable nodes connected to check node m, where $0 \le n \le N-1$, and $0 \le m \le M-1$. C(n)\m refers to exclusion of m from set C(n), and similarly V(m)\n refers to exclusion of n from set V(m).

In the VNU, variable node "n" has messages $R_{m'n}$ coming in from all check nodes m′ connected to it and its channel $L_{ch}(n)$. Hence, the outgoing message ("extrinsic") Q_{nm} on an edge n→m is the sum of all messages except R_{mn}. at iteration i, each variable node "n" calculates messages $Q_{nm}(i)$, which is sent from variable node "n" to each check node m∈C(n). Message $Q_{nm}(i)$ is the LLR of variable node "n" based on all check nodes in C(n)\m, and is calculated as defined by [7]

$$Q_{nm}(i) = L_{ch}(n) + \sum_{m' \in c(n)\backslash m} R_{m'n}(i-1) \quad (7)$$

where $L_{ch}(n)$ is the channel LLR of variable node "n". The computation is shown in Fig. 4-a, assuming that the variable node has degree = 3. The a posteriori LLR for a variable node is obtained by adding all the incoming messages at the variable node.

where "m" can be any check node in C(n). The above expression indicates that the variable-to-check message $Q_{nm}(i)$ in a current iteration can be directly calculated from the previous iteration and the check-to-variable message $R_{mn}(i-1)$ on the same edge from the previous iteration.

$$\sum_{n' \in v(m)} \oplus x_{n'} = 0 \quad (8)$$

$$x_n = \sum_{n' \in v(m)\backslash n} \oplus x_{n'} \quad (9)$$

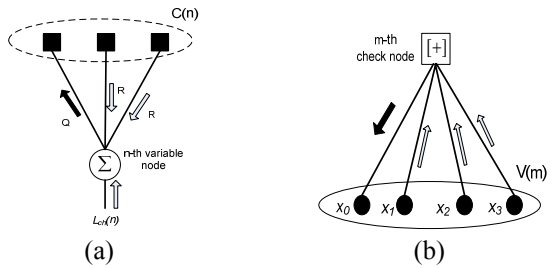

Figure 4. Operations in a belief propagation decoder: (a) Variable Node Update for a degree-3 variable node and (b) Check Node Update for a degree-4 check node. Only message update on one edge (marked with solid arrow) is illustrated but the similar operations are used to update messages on all the edges.

 ISBN: 978-1-61208-194-6

In the CNU, check node "m" has messages $Q_{n'm}$ coming in from all variable nodes "n'" connected to it. Each check node "m" calculates messages $R_{mn}(i)$, which is sent from check node m to each variable node "n" $\in$ V(m). Message $R_{mn}(i)$ is the LLR of variable node "n" based on all the variable nodes in V(m)\n.

B. Scheduling

Scheduling involves communicating messages from one node to another as dictated by the edge connections in the bipartite graph. There are two typical schedules of belief propagation: flooding and layered schedule [9]. In flooding, the entire bipartite graph is flooded with messages that are passed back and forth along all the edges, as illustrated in Fig. 5-a. However, this 'flooding' increases the complexity especially for longer block-sizes when the number of edges becomes large.

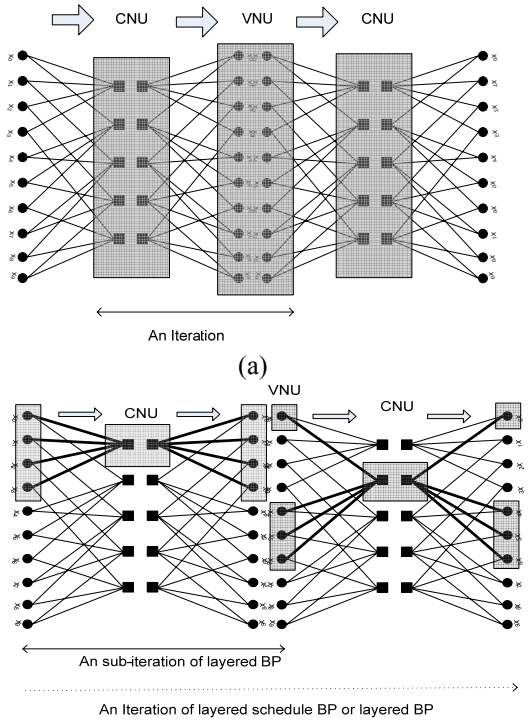

Figure 5. Message passing in the a) flooding schedule of the belief propagation algorithm, the shaded boxes indicate the CNU and VNU and block arrows indicate direction of message passing. Message passing occurs on a per-iteration basis b) layered BP, Shaded boxes indicate CNU and VNU and block arrows indicate direction of message passing. In the example, one CNU is done per sub-iteration. The edges that are updated in each sub-iteration are shown with thick solid lines.

In the layered algorithm, only a small fraction of the variable nodes and check nodes are updated per sub-iteration, as illustrated in Fig. 5-b. The messages generated in a sub-iteration of a current iteration are immediately used in subsequent sub-iterations within the same iteration. This leads to a faster flow of information and helps improve decoding speeds for the structured LDPC codes.

VI. LDPC COMPLEXITY CALCULATION

While actual decoding complexity depends on many factors such as hardware architecture, the decoding complexity as estimated in [9], based on operations count, the LDPC decoder computational complexity for Layered BP decoding per-iteration to be as follows

For the Check node update

$$(Nd_v + 2)(N - k) \qquad Additions$$

$$(2d_c - 3)(N - k) \qquad Comparisons$$

For the Variable node update

$$(N * d_v) \qquad Additions$$

So, the Total complexity/ iteration

$$(N - k)(2d_c + 1) + 2Nd_v \qquad operations$$

Total complexity

$$K\left(\frac{1}{R} - 1\right) + (4d_c + 1) \qquad (10)$$

where R=K/N is the code rate, N is the decoded frame length, d_c is the average row weight and d_v is the average column weight,

Table II uses the form of equation (10) which manifests the complexity as a function of code rates for different code rates per frame length and 20 iterations.

TABLE II. COMPLEXITY OF LDPC CODES FOR DIFFERENT CODE RATES FOR FRAME LENGTH "K"

Rate increasing ↓ / Complexity increasing ↑

Code rate	d_v and d_c values	Complexity/ iteration/ frame length	For 20 iterations/frame length K
Rate 1/3	dv = 4, dc = 6	2K (4dc+1) = 50 K	50 * 20* K = 1000K
Rate 1/2	dv = 3,dc = 6	K (4dc+1) = 25 K	25* 20* K = 500 K
Rate 3/4	dv = 3,dc = 12	0.3K (4dc+1)= 16.3 K	16.3* 20*K = 326 K

From Table II, it is concluded that the complexity is decreased as the code rate increased.

VII. LDPC AND TC PERFORMANCE COMPARISON

For the performance of these codes, a simulation is made for 7/8 turbo code and compared with the performance of QC-LDPC 7/8 (8176, 7156) in [10] and the result of comparison is depicted in Fig. 6. The results show that the performance of LDPC is better than the turbo code for higher rates. Another simulation is made for turbo code rate 1/2 and compared with the performance of LDPC rate 1/2 in [11] and the result of comparison is depicted in Fig. 7. The results show that the performance of turbo code is better than the LDPC for moderate code rates. LDPC is better in performance for high code rates (rate 7/8) than the Turbo Code, Beside that the iterations in LDPC can be done in parallel but for turbo code is in serial. Here, the Turbo code is better in performance for moderate code rate than the LDPC.

 ISBN: 978-1-61208-194-6

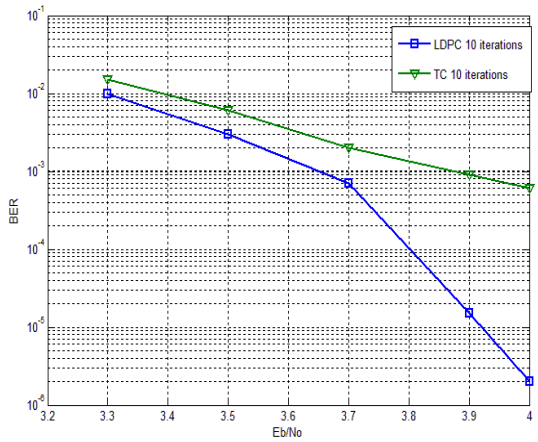

Figure 6. Performance comparison between Turbo code (O) and LDPC (□) for rate 7/8

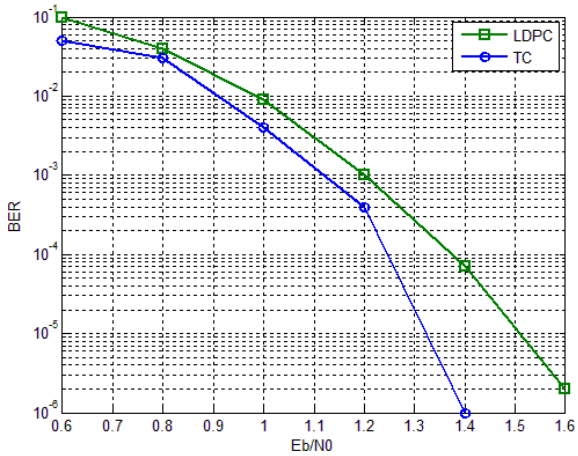

Figure 7. Performance comparison between Turbo code(O) and LDPC (□) for rate 1/2 and coded frame length 4000 bit

So, it is concluded that from the point of view of performance, the LDPC is recommended for higher code rates for communication systems applications, while in the low code rates it is better to use the turbo code. In the next section, a complexity comparison for turbo code and LDPC for different code rates will take place.

VIII. LDPC AND TC COMPLEXITY COMPARISON

For the LDPC codes, the decoding converges within 15 to 20 iterations, while it is well-known that ten turbo-decoding iterations are sufficient for convergence. Therefore, for a fair comparison between LDPC and turbo-decoding algorithms, the number of iterations is chosen to be 20 and 10, respectively. also we have to notice that the LDPC is error detection and correction code, so, when we reach error-free frame before we reach the 20 iteration the decoding will stops, while in Turbo code the decoding has to continue to the total number of iteration even if no more improvement. The operations count of LDPC and turbo decoding algorithms are listed in Table III. The calculation of complexity for Turbo code and LDPC was calculated for different code rates. the complexity of the turbo code is constant and does not depend on the code rate because different rates comes from the puncturing of the mother code rate, while in the LDPC the complexity is based on the code rate, the higher the code rate the lower the complexity and vice versa.

The calculation is made for rates 1/3, 1/2, and 3/4. The summary of calculations is in Table III.

TABLE III. COMPLEXITY COMPARISON FOR LDPC AND TURBO CODE FOR DIFFERENT CODE RATES

	LDPC	**TC**	**Complexity reduction Ratio**
Number of Iterations	20	10	
Rate 1/3	50*20*K=1000 K	725*10*K= 7250K	13%
Rate 1/2	25*20*K=500K	7250K	7%
Rate 3/4	16.3*20*K= 326K	7250K	5%
	Complexity decreased for higher rates	Complexity is constant	

IX. COMPLEXITY VS. PERFORMANCE

The simulation results for Turbo code and LDPC are shown in Fig. 6 and 7 for different code rates (1/2 and 7/8), and the complexity calculation is tabulated in Table-III which has the complexities at different code rate. So, it is concluded that for higher code rates, LDPC has better performance and lower complexity, while for rate 1/2 the turbo code has better performance so it should be used even it is more complex because the performance is an important issue. The brief of recommendations for applications for different code rate is summarized in Table IV.

TABLE IV. RECOMMENDED DECODING ALGORITHM AT DIFFERENT CODE RATES FOR LDPC AND TURBO CODE

Code Rate	**Complexity/ Iteration**	**Recommended Coding Technique**
Low Code Rates	1/2, 1/3, 1/4, 1/6	Turbo code
High Code Rates	2/3, 3/4, 7/8	LDPC

X. CONCLUSION AND FUTURE WORK

In this paper, a complexity needed to decode a Turbo code and LDPC were calculated; besides, the simulation of their performance was made. A comparison between two codes should compare the complexity and performance before applied in any communication system for specific application. The performance comparison of turbo code and LDPC were computed for rates 1/2 and 7/8. And the complexity for the two codes were calculated for code rates (1/2, 1/3, 3/4). The performance and complexity were based on ten decoding iterations for turbo code, while it is 20 iterations for LDPC.

It is concluded that the turbo code has better performance in moderate code rate (Rate 1/2) while the LDPC is recommended for higher code rates (3/4,7/8) because it has better performance beside less complexity compared with

 ISBN: 978-1-61208-194-6

turbo code. For turbo code, all code rates require the same decoding complexity since all code rates are obtained from the mother code via puncturing. In contrast, the LDPC decoding complexity decreases as the code rate increases.

REFERENCES

[1] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding:Turbo Codes", IEEE Transactions on Communications, vol. 44, pp. 1261-1271, Oct. 1996.

[2] R. G. Gallager, "Low Density Parity Check Codes," Monograph, M.I.T. Press, 1963.

[3] D. MacKay and R. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," Electronics Letters, July 1996

[4] K. Fagervik and S. Larssen, "Performance and Complexity Comparison of Low Density Parity Check Codes and Turbo Codes," Stravanger University Website.

[5] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. on Information Theory, vol. 20, pp. 284-287, March 1974.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding Turbo-codes," in IEEE International Conference on Communications, Geneva, Switzerland, May 1993, pp. 1064-1070.

[7] S. Lin and D. Castello, "Error Control Coding" 2nd ed. New Jersey: Pearson Prentice Hall, Chapter 16 and 17, 2004.

[8] T. Kwon, J. Sondeen and J. Draper, "Floating-Point division and square root implementation using a Taylor-series expansion algorithm with reduced look-up tables," in Proceedings of the IEEE Symp. Circuits and Systems, 2008, pp. 954-957.

[9] Y. Blankenship, Stephen Kuffner "LDPC Decoding for 802.22 Standard" IEEE P802.22, 2007.

[10] CCSDS , "Low Density Parity Check Codes for Use in Near –Earth and deep space communications" 131.1-O-2, 2007.

[11] Michael Yang, William E. Ryan and Yan Li, "Design of Efficiently Encodable Moderate-Length High-Rate Irregular LDPC Codes" IEEE Transaction on Communication, Vol. 52, No. 4, April 2004.

 ISBN: 978-1-61208-194-6