Chapter 8

Low-Density Parity- Check Codes

References

1. R.G. Gallager ," Low Density Parity Check Codes," IRE Trans. Inform. Theory , IT-8, pp.21-28 , 1962 .
2. R.M. Tanner, " A Recursive Approach to Low Complexity Codes ," IEEE Trans. Inform. Theory , IT -27 ,pp.533- 547 , Sep. 1981.
3. D.J.C. MacKay and R.M. Neal, " Good Codes Based on Very Sparse Matrices , " in " Cryptograph and Coding" , 5th IMA Conf. 1995, C.Boyd , Ed. ,Springer ,1995 , vol. 1025 , pp.100-111.
4. D.J.C. MacKay and R.M. Neal, " Near Shannon Limit Performance of Low Density Parity Check Codes ", Electronics Lett. Mar.1997, vol.33, no.6, pp.457-458 .
5. T. J. Richardson and R. Urbanke , " Efficient Encoding of Low-Density Parity- Check Codes ," IEEE Trans. Inform. Theory , pp.638-656 , Feb. 2001 .
6. S.-Y.Chung , G.D. Forney , Jr. , T. J. Richardson and R. Urbanke ,"On the Design of Low-density Parity- Check Codes within 0.0045 dB of the Shannon Limit , " IEEE Commun. Lett. , vol.5 , Feb. 2001 , pp.58-60
7. T.J. Richardson and M. Amin Shokrollahi and Rüdiger L. Urbanke , "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes, " IEEE Trans. Inform. Theory, 47(2), February 2001 , pp.673-680.
8. S. J . Johnson ,Iterative Error Correction, Cambridge University Press, 2010
9. Bernhard M.J. Leiner, " LDPC Codes – a brief Tutorial , April, 2005 (can be downloaded from Google)

8.1 Introduction

- LDPC codes were first discovered by R.G. Gallager in 1962.
  These codes have performance exceeding , in some cases, that of turbo codes with iterative decoding algorithms which are easy to implement, and are also parallelizable in hardware.
- However, LDPC codes have a significantly higher encode complexity than the turbo codes. Also, decoding of LDPC codes may require many more iterations than turbo decoding which means longer latency.

- Gallager’s remarkable discovery was mostly ignored by communications community researchers for almost 20 years.
  In 1981 , R. Tanner presented a new interpretation of the LDPC codes from a graphical point of view. Tanner’s work was also ignored by the coding theorists for another 14 years until the late 1990s.

- D.J. C.MacKay and R.M.Neal rediscovered LDPC codes in 1995 (after the turbo codes were introduced), and generated great interest and activity on the subject.

- Richardson and Urbanke in 2001 demonstrated that by using back-substitution , one can build encoders for most LDPC codes with complexity that grows almost linearly in block length.

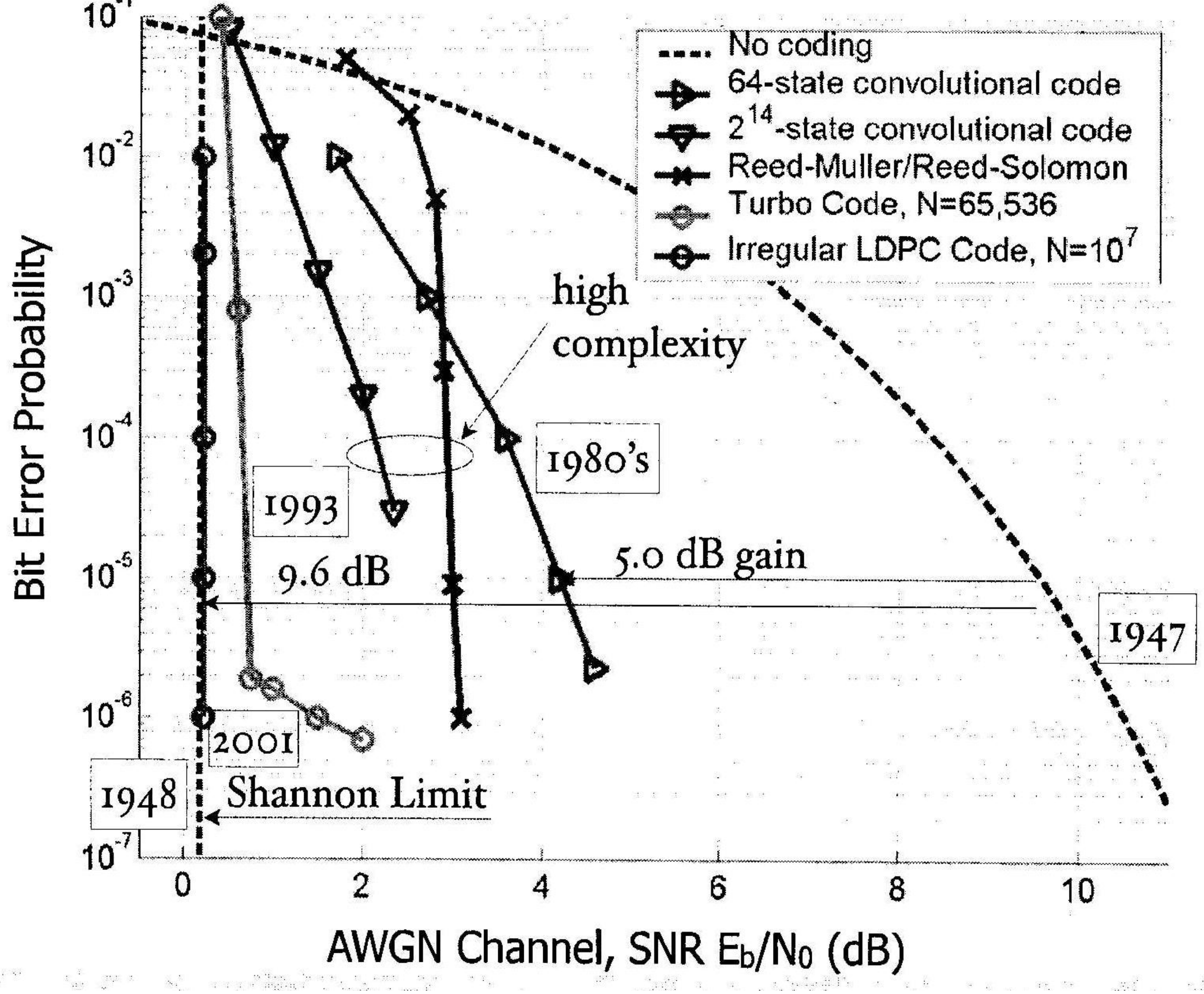
- **The standard iterative decoding algorithm uses a two-pass message-passing algorithm , proposed by Gallager in his Ph.D. thesis.**
- **The feature of LDPC codes to perform near the Shannon limit of a channel exists only for large block lengths . For example, there have been simulations of irregular codes that perform within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} with a block length of 10^{7} (S. Chung et al.) .**

- **The LDPC codes have been adopted in the second-generation digital video broadcasting (DVB-S2) via satellite, Wireless LAN (IEEE 802.11n), Wireless MAN (IEEE 802.16m), mobile broadband wireless access (MBWA) network (IEEE 802.20) , and advanced magnetic and magneto-optic storage/recording systems.**

- Although implementation of LDPC codes has lagged that of other codes, notably the turbo codes, the absence of encumbering software patents has made LDPC attractive to some.
- In 2003, an LDPC code beat six turbo codes to become the error correcting code in the new DVB-S2 standard for the satellite transmission of digital television .

- In 2008, LDPC beat Convolutional Turbo Codes as the FEC scheme for the ITU-T G.hn standard.
- G.hn chose LDPC over Turbo Codes because of its lower decoding complexity (especially when operating at data rates close to 1 Gbit /s) .
- LDPC is also used for 10GBase-T Ethernet, which sends data at 10 gigabits per second over twisted pair cables.

Reducing the Gap to Capacity. Rate R=1/2 Codes



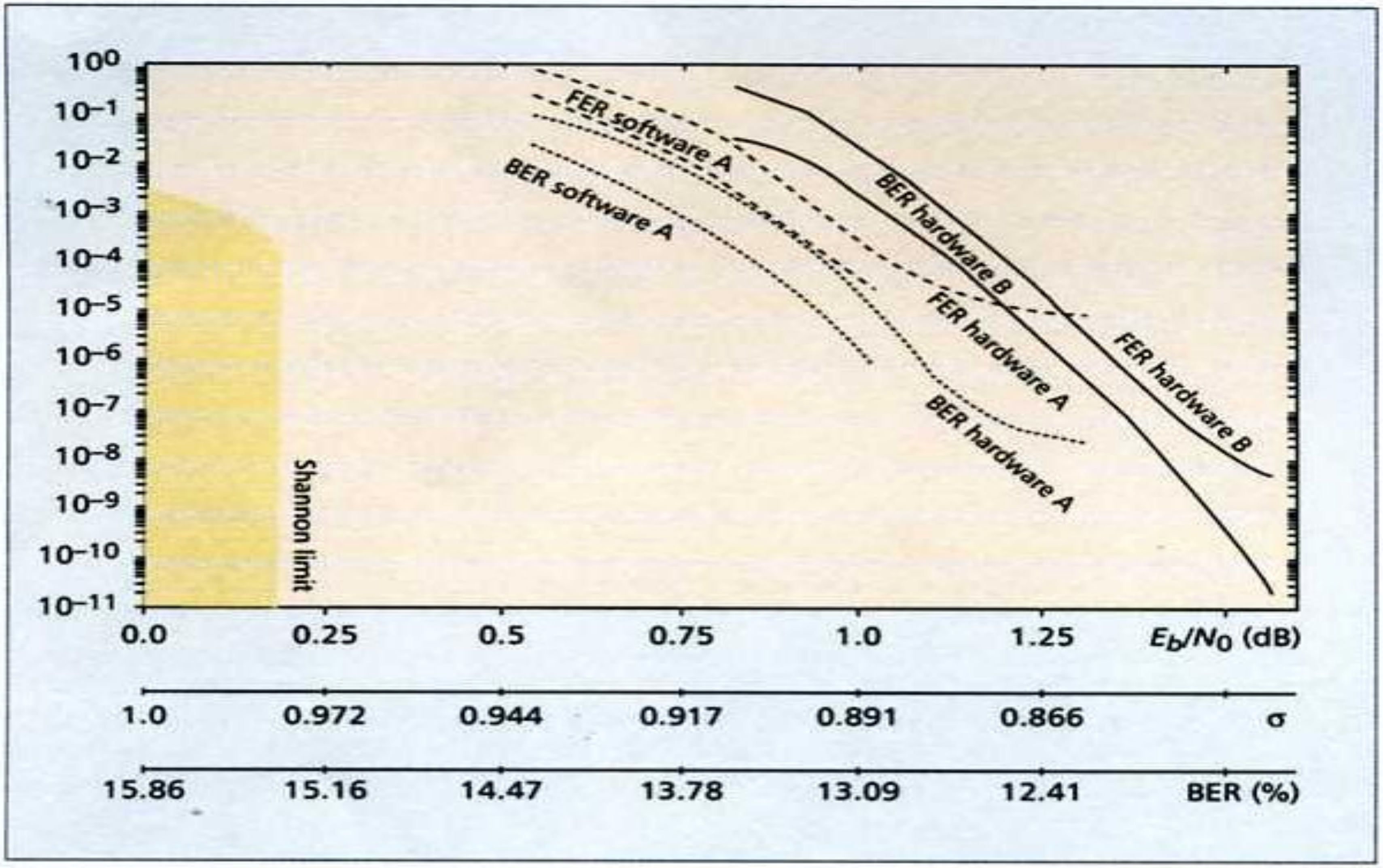


Figure 5. *FER and BER curves for rate 1/2 LDPC codes over the AWGN decoded using belief propagation and a 5-bit message passing algorithm. Code B was designed for deeper error floor, illustrating the trade-off.*

8.2 LDPC Code and Tanner Graph

- **LDPC codes are linear block codes defined by a sparse parity – check matrix H . The set of valid *n*-bit codewords**

  v is defined by $\mathbf{H} . \mathbf{v}^T = 0$ (8.1)

  where $\mathbf{v} = (v_0 , v_1 , \ldots , v_{n-1})$

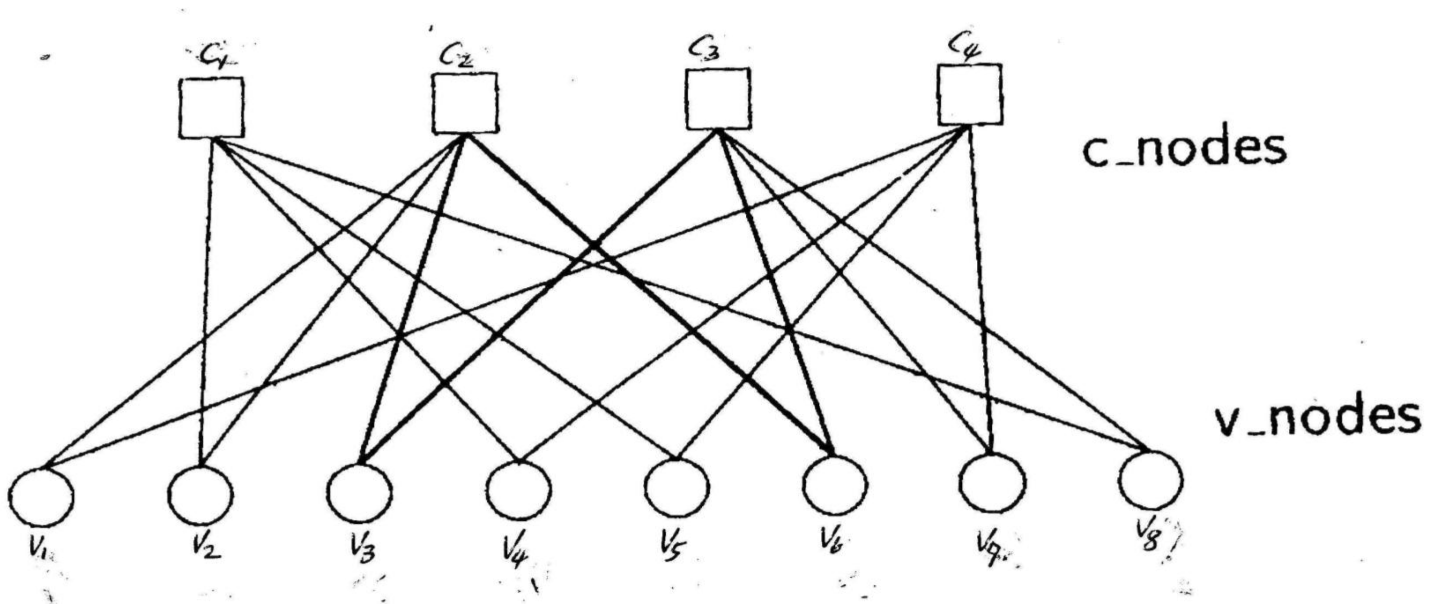
- **Tanner introduced an effective graphical representation for LDPC codes. The parity-check matrix H can be efficiently represented a bipartite (Tanner) graph , as shown in Fig.8.1 .**

  The graphs not only provide a complete representation of the codes, they also help to describe the decoding algorithm.

- **In bipartite graphs , the nodes in these graphs are separated into two distinct types , and edges are only connecting nodes of two different types. These two types of nodes in Tanner graphs are denoted as variable-nodes (V- nodes) and check-nodes (C- nodes).**
- **Fig. 1 is an example of Tanner graph for a (8, 4) code . The corresponding parity-check matrix is given by**

$$
\mathbf{H} = \begin{matrix} \mathbf{0\ 1\ 0\ 1\ 1\ 0\ 0\ 1} \\ \mathbf{1\ 1\ 1\ 0\ 0\ 1\ 0\ 0} \\ \mathbf{0\ 0\ 1\ 0\ 0\ 1\ 1\ 1} \\ \mathbf{1\ 0\ 0\ 1\ 1\ 0\ 1\ 0} \end{matrix}
$$

Fig.8.1



In this example , the parity-check equations are expressed by

At c_1 $v_2 + v_4 + v_5 + v_8 = 0$

At c_2 $v_1 + v_2 + v_3 + v_6 = 0$

At c_3 $v3 + v_6 + v_7 + v_8 = 0$

At c_4 $v_1 + v_4 + v_5 + v7 = 0$

- **This Tanner graph consists of *m* C- nodes (the number of check bits) and *n* V- nodes (the number of bits in a codeword). Check-node z_i is connected to variable node c_j if the element h_{ij} of H is a *1* .**

  The check nodes correspond to the row of H. The edges in the Tanner graph correspond to the 1s in H .

  Cycle : In a graph , a cycle is a path that starts from a node *i* and ends in i .

  Girth : The girth of a graph is the smallest cycle in that graph.

- **Regular LDPC codes**

  A LDPC code is called regular if the weight w_c of each column of H is a constant and the weight w_r of each row is also a constant , where $w_r = w_c (n/m)$ .For the example in Fig.1 , $w_c = 2$, $w_r = 4$

- **M.G.Luby *et al* in 1998 demonstrated that LDPC codes based on *i*rregular graphs can substantially outperform similar codes based on regular graphs.**
- **Complexity in iterative decoding has three parts :**

  (1) the complexity of the local computations

  (2) the complexity of the interconnection (i.e., the routing of information)

  (3) the number of iterations : the number of times the local computation need to be repeated.

8.3 Construction of LDPC Codes

- For large block sizes, LDPC codes are commonly constructed by first studying the behavior of decoders. As the block-size tends to infinity, LDPC decoders can be shown to have a noise threshold below which decoding is reliably achieved, and above which decoding is not achieved. The construction of a specific LDPC code after this optimization falls into two main types of techniques:

  (a) Pseudo-random techniques

  (b) Combinatorial approaches

  (c) Finite geometry approach

- **A random construction constructs LDPC codes by a pseudo-random approach based on theoretical results. For large block-size, a random construction gives good decoding performance. In general, pseudo-random codes have complex encoders, however pseudo-random codes with the best decoders also can have simple encoders. Various constraints are often applied to help the good properties expected at the theoretical limit of infinite block size to occur at a finite block size.**
- **Combinatorial approaches can be used to optimize properties of small block-size LDPC codes or create codes with simple encoders.**
- **One more way of constructing LDPC codes is to use finite geometries. This method was proposed by Y. Kou *et al.* in 2001.**

Encoding of Regular LDPC Codes

- **An (n ,k) LDPC code can be generated by an (*n-k*) x *n* parity-check matrix H .**
- **The parity-check matrix H may be expressed as**

$$H = [\ A_1^T \quad A_2^T\ ]$$

were A_1 is a k x (n-k) matrix , and A_2 is an (n-k)x (n-k) matrix.

Then the corresponding generator matrix of the LDPC code is given by

$$G = [\ 1_{kxk} \quad A_1 A_2^{-1}\ ]$$

8.4 Iterative Decoding of LDPC Codes

8.4.1 Principle of Iterative Decoding

- Codes are constructed so that the relationship between their bits is locally simple , admitting simple local decoding . The local description of the codes are interconnected in a complex (e.g. random –like) manner , introducing long-range relationships between the bits . Relatively high global description complexity is thereby introduced in the interconnection between the simple local structures .
- Iterative decoding proceeds by performing the simple local decoding and then exchanging the results , passing messages between locals across the " complex " interconnection . The locals repeat their simple decodings , taking into account the new information provided to them from other locals. Tanner graph can be used to represent this process. Locals are nodes in the graph , and interconnections are represented as edges .

8.4.2 Iterative Decoding Algorithms

- The class of algorithms used to decode LDPC codes are collectively termed *message –passing algorithm* , since their operation depends passing of messages along the edges of the Tanner graph describing the LDPC code
- Each Tanner graph node works in isolation, having access only to the messages on the edges connected to it
- In message-passing decoders , messages are exchanged along the edges of the graph, and computations are performed at the nodes. Each message represents an estimate of the bit associated with the edge carrying the message . Each variable node in the decoder gets to see the bit that arrived at the receiver corresponding to the one that was transmitted from the equivalent node at the transmitter.

- **The messages pass back and forth between the variable nodes and check nodes iteratively until a result is obtained (or the process is halted) .**

- **Two most popular message-passing algorithms are**
  (a) bit-flipping decoding and
  (b) belief-propagation (or sum-product) decoding .

In bit-flipping decoding , the message are binary .
A bit-flipping algorithm can be viewed as a hard-decision message-passing algorithm for LDPC codes.
In belief-propagation decoding , the message are probabilities (or their log-likelihood ratio) that represent a level of belief about the value of the codeword bits .

8.4.3 Bit-Flipping Decoding

- **For the bit-flipping algorithm , the *m*th c-node determines its decision on the *i*th V-node by assuming that the *n*th bit has been erased and choosing the value 1 or 0 that satisfies the *m*th parity-check equation. The *j*th C-node thus determines a value for the nth bit that is completely independent of the value for the nth bit just received by it. The C-node is said to be creating extra, extrinsic , information about the *n*th bit.**
- **At the variable node v_i , all the extrinsic information about a bit is compared with the information received from the channel to determine the most likely bit value.**
- **If the majority of the messages received by a variable node v_i are different from its received value , the variable node changes (flips) its current value .**
- **The process is repeated until all parity-check equations are satisfied (or a maximum number of iterations has passed).**

- **The iterative process can be described as follows.**

Step 1 V-node v_i send a message to their C- nodes c_j .

In the first round , v_i only has the received bit y_i .

Step *2* *C*-nodes c_j determine a response to every connected variable nodes . *The response message contains the bit that* c_j believes to be the correct *one for this* V-node v_i , assuming that the other V-nodes connected to c_j *are correct.*

The LDPC decoder might find out that the received bits are correct and terminates the decoding if all parity-check equations are fulfilled .

Step 3 Each V-node receives these responses from C-nodes and use this information along with the received bit to find out that the originally received bit is correct or not .

Step 4 go to Step 2.

Example

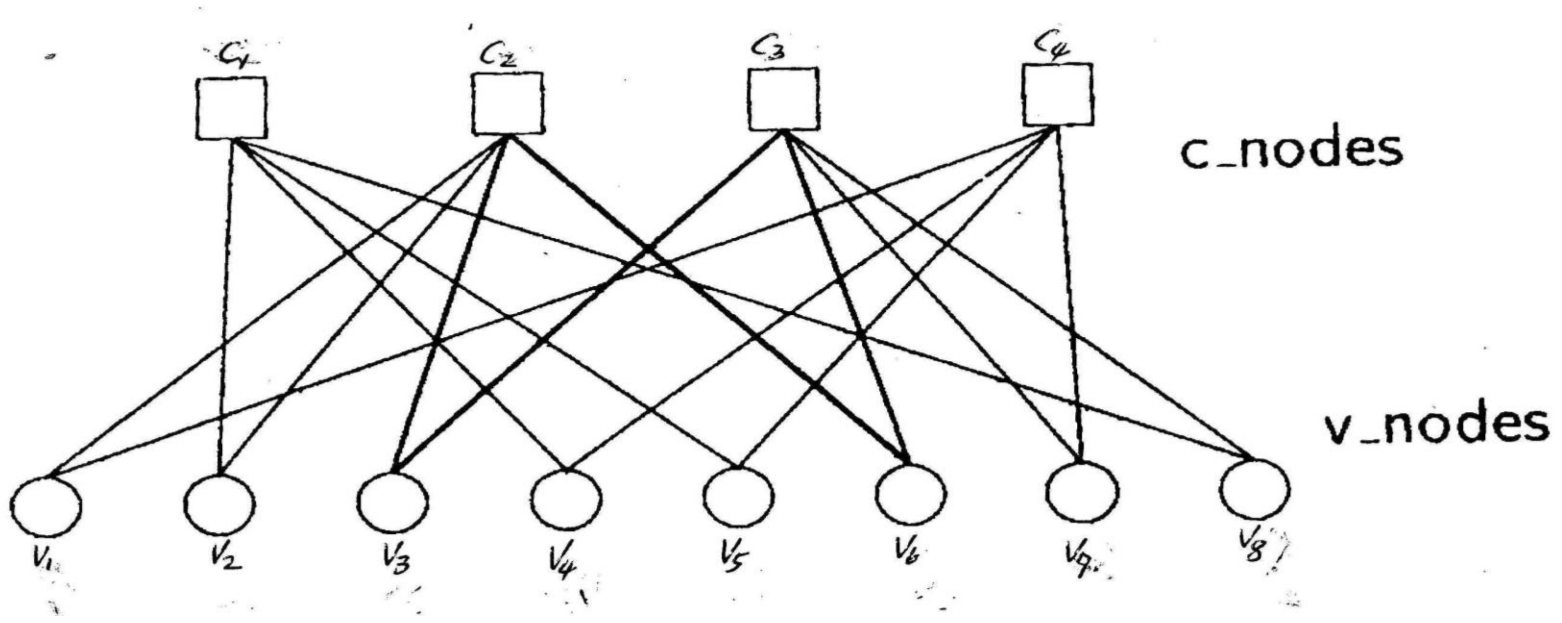
In this example , the parity-check equations are expressed by

At c_1 $v_2 + v_4 + v_5 + v_8 = 0$

At c_2 $v_1 + v_2 + v_3 + v_6 = 0$

At c_3 $v3 + v_6 + v_7 + v_8 = 0$

At c_4 $v_1 + v_4 + v_5 + v_7 = 0$



The received codeword is 1 1 0 1 0 1 0 1
Message received and sent by the C-nodes in Step 2 are given in the following table .

Check nodes	Received / Sent
c_1	**Received** v_2 -->1 , v_4 -->1 , v_5 -->0 , v_8-->1
	Sent 0 --> v_2 , 0 --> v_{44} , 1 --> v_5 , 0 --> v_8
c_2	**Received** v_1 -->1 , v_2 -->1 , v_3 -->0 , v_6 -->1
	Sent 0 --> v_1 , 0 --> v_2 , 1 --> v_3 , 0 --> v_6
c_3	**Received** v_3 -->0 , v_6 -->1 , v_7-->0 , v_8-->1
	Sent 0 --> v_3 , 1 --> v_6 , 0 --> v_7 , 1 --> v_8
c_4	**Received** v_1-->1 , v_4 -->1 , v_5-->0 , v_7 -->0
	Sent 1 --> v_1 , 1 --> v_4 , 0 --> v_5 , 0 --> v_7

In Step 3 of the decoding algorithm , each v-node has three sources of information concerning its bit , the original bit received and two suggestions from the check nodes. Majority vote is used to make decision , as show in the following table..

V-node	y_i received	Messages from check–node		Decision
v_1	1	$c_2 \rightarrow 0$	$c_4 \rightarrow 1$	1
v_2	1	$c_2 \rightarrow 0$	$c_4 \rightarrow 0$	0
v_3	0	$c_2 \rightarrow 1$	$c_3 \rightarrow 0$	0
v_4	1	$c_2 \rightarrow 0$	$c_4 \rightarrow 1$	1
v_5	0	$c_1 \rightarrow 1$	$c_4 \rightarrow 0$	0
v_6	1	$c_2 \rightarrow 0$	$c_3 \rightarrow 1$	1
v_7	0	$c_3 \rightarrow 0$	$c_4 \rightarrow 0$	0
v_8	1	$c_1 \rightarrow 1$	$c_3 \rightarrow 1$	1

In this example, the second execution of Step 2 would terminate the decoding process since v_1 has voted for 0 in the last step . This corrects the transmission error and all check equations are now satisfied. Note that a **bit-flipping algorithm** is the name given to hard decision message-passing algorithm for LDPC codes.

8.4.4 Belief Propagation Algorithm

- The belief propagation (BP) algorithm , also known as Sum-Product algorithm) ,was presented in Gallager's work. The message passed along the edges in the Tanner graph are probabilities , or beliefs .

- **Before presenting the algorithm , some notations will be introduced as follows.**

  (a) Conditional probability $p_i = p\,(v_i=1 \mid y_i)$ **(8.2)**

  (b) q_{ij} **is a message sent by the variable nodes** v_i **to the check node** c_j **.**

  Every message contains always the pair q_{ij} **(0) and** q_{ij} **(1) which stands for the amount of belief that** x_i **is a "0" or a "1" .**

  (c) r_{ji} **is a message (extrinsic information) sent by the check node** c_j **to the variable node** v_i **.**

  Again , there are a r_{ji}**(0) and a** r_{ji} **(1) to indicate the current amount of belief in that** y_i **is a "0" or a "1" , respectively.**

- **SPA**
- **At the beginning , all variable nodes send their q_{ij} messages to C-nodes . Since no other information is available at this step ,**

$$q_{ij}(1) = p_i$$

and $$q_{ij}(0) = 1 - p_i ,$$

Then the check nodes calculate their response messages r_{jij} :

$$r_{ji}(0) = \tfrac{1}{2} + \tfrac{1}{2} \prod_{i' \varepsilon V_{j \setminus i}} (1 - 2\, q_{i'j}(1)) \qquad (8.3)$$

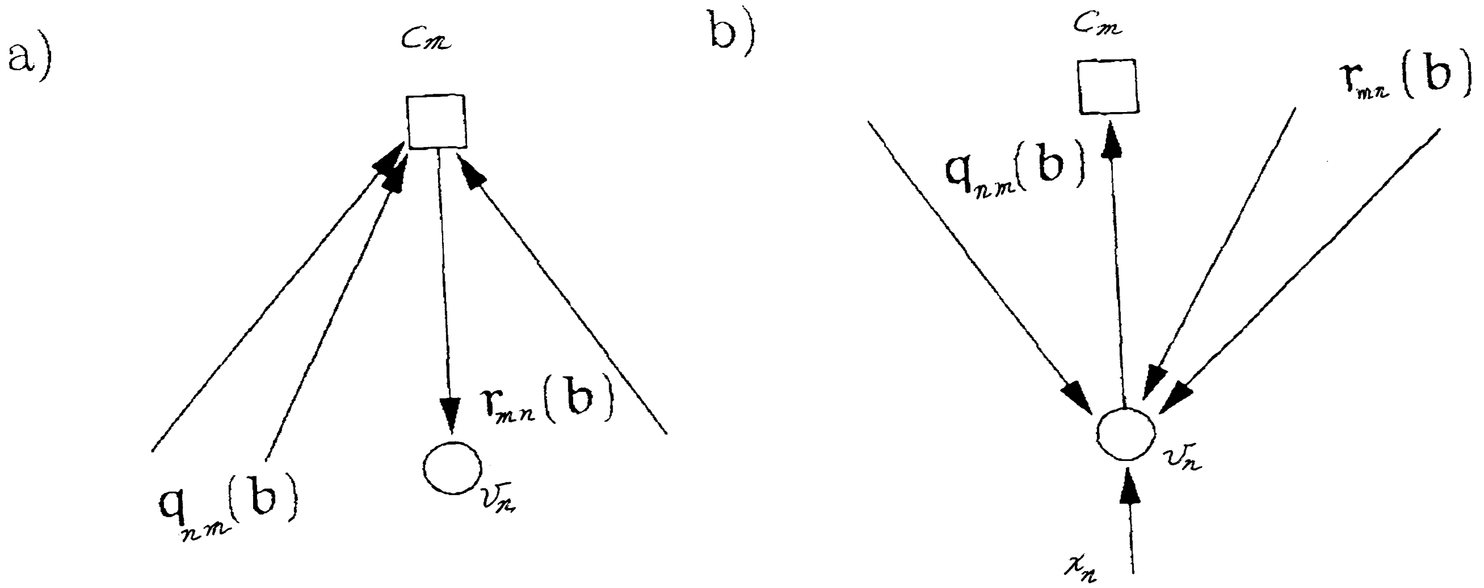
and $$r_{ji}(1) = 1 - r_{ji}(0) \qquad (8.4)$$

where $V_{j \setminus i}$ means all V-nodes except v_i. .
Note that $r_{ji}(0)$ is basically the probability that there is an even number of *1*s among $V_{j \setminus i}$.

Remarks : Eq. (8.3) uses the following result from Gallager .

Lemma : For a sequence of *K* independent binary digits a_i with an probability of p_i for $a_i = 1$, the probability that the whole sequence contains an even number of *1*'s is

$$\tfrac{1}{2} + \tfrac{1}{2} \prod_{i=1}^{K} (1 - 2 p_i)$$



Illustrates the calculation of $r_{ji}(b)$ and $q_{ij}(b)$

Next ,the V-nodes update their response messages to the check- nodes. This is done according to following equations ,

$$q_{ij}(0) = K_{ij}\,(1 - p_i) \prod_{j' \,\varepsilon\, C_{i \setminus j}} r_{j'i}(0)$$

$$q_{ij}(1) = K_{ij}\,p_i \prod_{j' \,\varepsilon\, C_{i \setminus j}} r_{j'i}(0)$$

where $C_{i \setminus j}$ means all C-nodes except c_j .
The constants K_{ij} are chosen to ensure that

$$q_{ij}(0) + q_{ij}(1) = 1$$

Also , at this step V-nodes update the decision v_n with information from every C-nodes .
If the estimated v satisfies Hv =0 , then the algorithm terminates.

In practical computations , we perform the algorithm in log-domain . Denote that the APP log-likelihood ratio is expressed by

$$R_i = \ln\ [\, p\,(\, v_i=1 \mid y_i) \;/\; p\,(\, v_i= 0 \mid y_i) \,]$$

(a) For BSC ,

$$R_i = \begin{cases} \ln \varepsilon/(1-\varepsilon) & \text{if } y_i = 0 \\ \ln (1-\varepsilon)/\varepsilon & \text{if } y_i = 1 \end{cases}$$

***(b)* For AWGN channel ,**

$$R_i = (2 / \sigma_n^2)\, v_i$$

- **In the log-domain , we can expressed the extrinsic information from C- node *j* to V-node *i* as**

$$R_{ji} = \ln\,[r_{ji}(1) / r_{ji}(0)] = \ln \left\{ \left[\tfrac{1}{2} - \tfrac{1}{2} \prod_{i' \varepsilon V j \setminus i} (1-2\, q_{i'j}(1)) \right] \Big/ \left[\tfrac{1}{2} + \tfrac{1}{2} \prod_{i' \varepsilon V j \setminus i} (1-2\, q_{i'j}(1)) \right] \right.$$

- **By defining a LLR measure $Q_{i'j}$ as**

$$Q_{i'j} = \ln q_{i'j} / (1-q_{i'j}) \qquad (8.\ )$$

and using the relationship

$$\tanh [\, ½ \ln (1-p)/p \,] = 1-2p \ , p< 1 .$$

R_i can be expressed as

$$R_{ji} = \ln \{ [\, 1 - \prod_{i' \varepsilon Vj \setminus i} (1- e^{-Q_{i'j}})/ (1+ e^{-Q_{i'j}}) \,] \quad / \quad [1 + \prod_{i' \varepsilon Vj \setminus i} (1- e^{-Q_{i'j}})/ (1+ e^{-Q_{i'j}})] \}$$

$$= \ln \{ [\, 1 - \prod_{i' \varepsilon Vj \setminus i} \tanh (Q_{i'j} /2) \,] \ / \ [1 + \prod_{i' \varepsilon Vj \setminus i} \tanh (Q_{i'j} /2) \ \} \qquad (8.\ )$$

Alternatively , using the relationship

$$2 \tanh^{-1} p = \ln (1+p)/(1-p) \,,$$

the extrinsic information can be expressed as

$$R_{ji} = -2 \tanh^{-1} \prod_{i' \varepsilon V_j \backslash i} \tanh (Q_{i'j} / 2) \qquad \text{(8.)}$$

- **Each variable node has access to the input LLR , R_i, and to the LLR from every connected check node .**
  The total LLR of the *i*th bit is the sum of these LLRs :

$$L_i = R_i + \Sigma_j \; R_{ji}$$

The message sent from the *i*th V-node to the *j*th C-node is the sum of L_i without the component R_{ji} just received from the *j*-th C-node :

$$Q_{ij} = \sum_{j' \varepsilon C_i \backslash j} R_{j'i} + R_i$$

Estimation of the coded bit :

$$\hat{v_i} = \begin{cases} 1 & L_i \geqq 0 \\ 0 & L_i < 0 \end{cases}$$

Summry : Decoding Procedures

- **Step 0 (initial condition)**

  Initially , the inputs to the decoder are the log likelihood ratios for the *a priori* message probabilities from each channel . That is,

  $R_{ji} = R_i \qquad j = 1,2,..,m \qquad i = 1,2,\ldots, n$

  $Q_{ij} = \ln [p_i / (1- p_i)]$

Step 1

Compute

$$R_{ji} = \ln \left\{ \left[1 - \prod_{i' \varepsilon V_j \setminus i} \tanh (Q_{i'j}/2) \right] \Big/ \left[1 + \prod_{i' \varepsilon V_j \setminus i} \tanh (Q_{i'j}/2) \right] \right\} \qquad (8.\ )$$

$j = 1,2,..,m \qquad i = 1,2,\ldots, n$

Step 2

Compute

$$L_i = \Sigma_j \; R_{ji} + R_i$$

$$\hat{v_i} = \begin{matrix} 1 & L_i \geqq 0 \\ 0 & L_i < 0 \end{matrix}$$

Step 3

Check if $H c^T = 0$ **or** $I = I_{max}$

If not , go to next step .

Step 4

Compute

$$Q_{ij} = \sum_{j' \varepsilon C_i \setminus j} R_{j'i} + R_i$$

and go to Step 1 and repeat the procedure.

Notes

- The renaissance of LDPC codes did not mark the end of the turbo codes . LDPC codes have performance and complexity advantages over turbo codes at high codes , but turbo codes are currently still the best solution for the lower code rates . This natural partition meant that the standard family of turbo codes at rates 1⁄ 6 , ¼ , 1⁄3 , ½ could live in harmony with a proposed standard of LDPC codes at rates ½ , 2⁄3 , 4⁄5 , and 7⁄ 8.

- $\tanh(x/2) = [(e^{x}-1)/(e^{x}+1)]$

Example: (Johnson , pp.65-67)

The LDPC code is used to encode a message sequence ,

The codeword from the encoder output is

c = [0 0 1 0 1 1]

The vector c is sent through a BSC with crossover probability ε= 0.2 ,

and the received signal is

y = [1 0 1 0 1 1]

For BSC $$R_i = \begin{cases} \ln \varepsilon/(1-\varepsilon) & \text{if } y_i = 0 \\ \ln (1-\varepsilon)/\varepsilon & \text{if } y_i = 1 \end{cases}$$

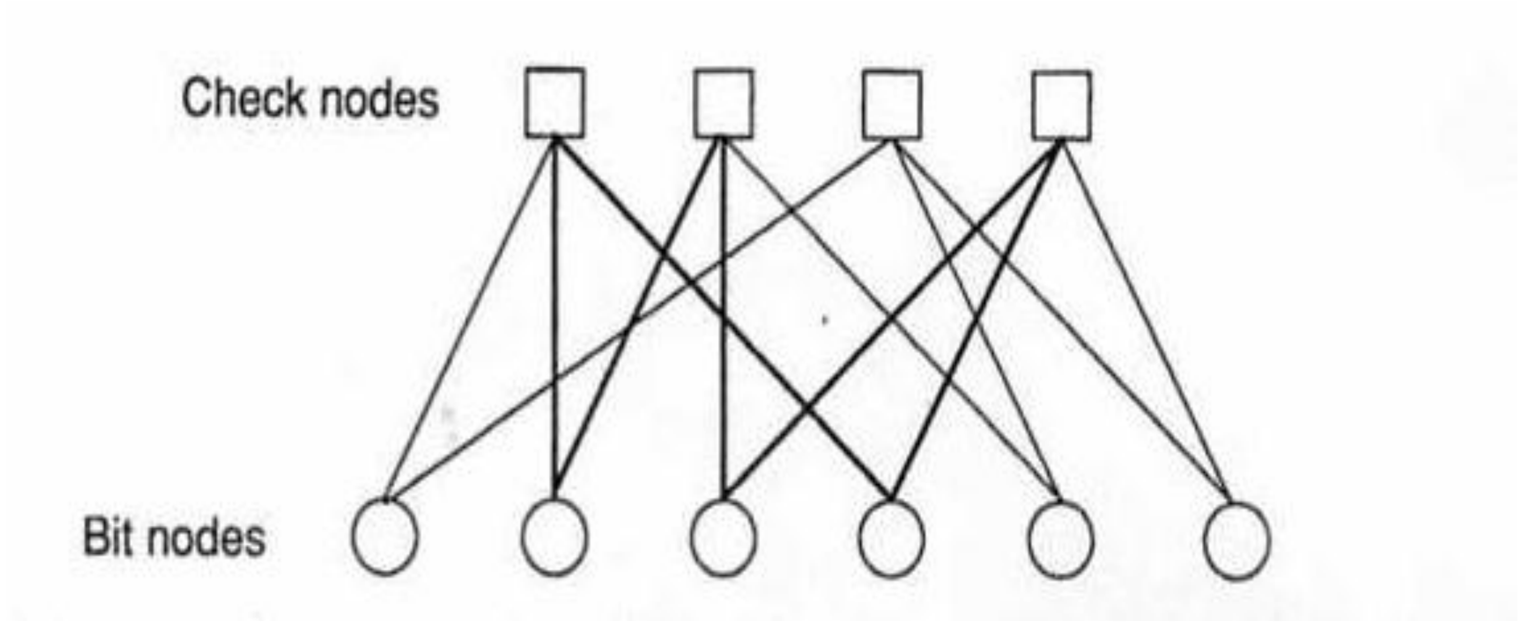
For this channel ε= 0.2

Thus, $\ln \varepsilon/(1-\varepsilon) = -1.3863$ if $y_i = 0$

$\ln (1-\varepsilon)/\varepsilon = 1.3863$ if $y_i = 1$

and then R = [1.3863 -1.3863 1.3863 - 1.3863 1.3863 1.3863]

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$



Decoding (Log-BP)

- **To begin the decoding we set the maximum number of iterations to 3 .**
- **At initialization ,** $Q_{ij} = R_i$

The first bit is included in the first and third checks and so Q_{11} *and* Q_{13} **are initialized to** R_1 **.**

$$Q_{11} = R_1 = 1.3863 \quad \text{and} \quad Q_{13} = R_1 = 1.3863$$

Repeating this for the remaining bits gives :

For $i = 2$ $Q_{21} = R_2 = -1.3863$ **and** $Q_{23} = R_2 = -1.3863$

For $i = 3$ $Q_{32} = R_3 = 1.3863$ **and** $Q_{34} = R_3 = 1.3863$

For $i = 4$ $Q_{41} = R_4 = -1.3863$ **and** $Q_{44} = R_4 = -1.3863$

For $i = 5$ $Q_{52} = R_5 = 1.3863$ **and** $Q_{53} = R_5 = 1.3863$

For $i = 6$ $Q_{63} = R_6 = 1.3863$ **and** $Q_{64} = R_6 = 1.3863$

- **Calculation of extrinsic probabilities for check-to-variable message passing**

The first parity-check includes the first, second and fourth bits and so the extrinsic probability from the first check node to the first variable node depends on the probabilities of the second and fourth bits :

$$R_{11} = \ln \{ [1 - \tanh (Q_{21} /2) \tanh (Q_{41} /2)] \; / \; [1+ \tanh (Q_{21} /2) \tanh (Q_{41} /2)] \}$$

$$= \ln \{ [1- \tanh (1.3863/2) \tanh(1.3863/2)] \; / \; [1+ \tanh (1.3863/2) \tanh(1.3863/2)] \}$$

$$= \ln \{ (1- 0.6 \times 0.6)/ (1 + 0.6\times0.6) \}$$

$$= - 0.7538$$

Similarly , $R_{12} = \ln \{ [1 - \tanh (Q_{11} /2) \tanh (Q_{41} /2)] \; / \; [1+ \tanh (Q_{11} /2) \tanh (Q_{41} /2)] \}$

$$= 0.7538$$

$$R_{14} = \ln \{ [1 - \tanh (Q_{11} /2) \tanh (Q_{21} /2)] \; / \; [1+ \tanh (Q_{11} /2) \tanh (Q_{21} /2)] \}$$

$$= 0.7538$$

Next, the second check node connects to the second , third and fifth bits and so the extrinsic probabilities are :

$$R_{22} = \ln \{ [1 - \tanh (Q_{32}/2) \tanh (Q_{52} /2)] \; / \; [1+ \tanh (Q_{32} /2) \tanh (Q_{52} /2)] \}$$

$$= - 0.7538$$

$$R_{23} = \ln \{ [1 - \tanh (Q_{22}/2) \tanh (Q_{52} /2)] \; / \; [1+ \tanh (Q_{22} /2) \tanh (Q_{52} /2)] \}$$

$$= 0.7538$$

$$R_{25} = \ln \{ [1 - \tanh (Q_{22}/2) \tanh (Q_{32} /2)] \; / \; [1+ \tanh (Q_{22} /2) \tanh (Q_{32} /2)] \}$$

$$= 0.7538$$

Repeating for all check , we obtain :

R_{31} = - 0.7538 R_{35} = - 0.7538 R_{36} = - 0.7538

R_{43} = 0.7538 R_{44} = - 0.7538 R_{46} = 0.7538

- **Check for a valid codeword :**

calculating the LLR for each bit, making a hard-decision , and checking the syndromes .

The total LLR for the first bit , L_1 , includes the extrinsic LLRs from the first and third check bits and an intrinsic LLR from the channel :

$$L_1 = R_1 + R_{11} + R_{31} = 1.3863 - 0.7538 - 0.7538 = -0.1213$$

Similarly , the total LLRs of other bits are

$$L_2 = R_2 + R_{21} + R_{22} = -1.3863$$

$$L_3 = R_3 + R_{23} + R_{43} = 2.8938$$

$$L_4 = R_4 + R_{14} + R_{44} = -1.3863$$

$$L_5 = R_5 + R_{25} + R_{35} = 1.3863$$

$$L_6 = R_6 + R_{36} + R_{46} = 1.3863$$

The estimated codeword is then given by

$$\hat{v} = (001011)$$

and the syndrome is

$$s = (\hat{v})^T H = (0000)$$

Thus , $\hat{v}$ = (001011) is the decode word.

Appendix :LDPC Code Encoder (IEEE 802.11n)

- For each of the three available codeword block lengths, the LDPC encoder supports rate 1/2, 2/3, 3/4 and 4/5 encoding. The LDPC encoder is systematic, which means it encodes an information block, $c=(i_0, i_1, \ldots\ldots, i_{k-l})$, of size k, into a codeword, c, of size n, $c=(i_0, i_1, \ldots\ldots, i_{(k-l)}), p_0, p_1, \ldots\ldots, p_{(n-k-1)}$), by adding (n-k) parity bits obtained so that $H*c^T=0$, where H is the parity-check matrix.

- Parity Check Matrices

- Each of the parity-check matrices can be partitioned into square sub-blocks (sub-matrices) of size $Z \times Z$. These sub-matrices are either cyclic-permutation of the identity matrix or null sub-matrices.
- The cyclic-permutation matrix P_i is obtained from the $Z \times Z$ identity matrix by cyclically shifting the columns to the right by i elements. The matrix P_0 is the $Z \times Z$ identity matrix.
- The parity check matrices for each kind of code length and code rate are shown below .
- The - entry means a null (all zero) block ,
- The 0 entry means an identity matrix

- **Table 1 Parity check matrix for codeword block length n= 648 bits, sub-block size is Z =27 bits . Code rate R = 1/2**

0	–	–	–	0	0	–	–	0	–	–	0	1	0	–	–	–	–	–	–	–	–	–	–
22	0	–	–	17	–	0	0	12	–	–	–	–	0	0	–	–	–	–	–	–	–	–	–
6	–	0	–	10	–	–	–	24	–	0	–	–	–	0	0	–	–	–	–	–	–	–	–
2	–	–	0	20	–	–	–	25	0	–	–	–	–	–	0	0	–	–	–	–	–	–	–
23	–	–	–	3	–	–	–	0	–	9	11	–	–	–	–	0	0	–	–	–	–	–	–
24	–	23	1	17	–	3	–	10	–	–	–	–	–	–	–	–	0	0	–	–	–	–	–
25	–	–	–	8	–	–	–	7	18	–	–	0	–	–	–	–	–	0	0	–	–	–	–
13	24	–	–	0	–	8	–	6	–	–	–	–	–	–	–	–	–	–	0	0	–	–	–
7	20	–	16	22	10	–	–	23	–	–	–	–	–	–	–	–	–	–	–	0	0	–	–
11	–	–	–	19	–	–	–	13	–	3	17	–	–	–	–	–	–	–	–	–	0	0	–
25	–	8	–	13	18	–	14	9	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
3	–	–	–	16	–	–	2	25	5	–	–	1	–	–	–	–	–	–	–	–	–	–	0

- **Table 2 Parity check matrix for codeword block length n=1944 bits, sub-block size is Z=81 bits. Code rate R = ½**

57	–	–	–	50	–	11	–	50	–	79	–	1	0	–	–	–	–	–	–	–	–	–	–
3	–	28	–	0	–	–	–	55	7	–	–	–	0	0	–	–	–	–	–	–	–	–	–
30	–	–	–	24	37	–	–	56	14	–	–	–	–	0	0	–	–	–	–	–	–	–	–
62	53	–	–	53	–	–	3	35	–	–	–	–	–	–	0	0	–	–	–	–	–	–	–
40	–	–	20	66	–	–	22	28	–	–	–	–	–	–	–	0	0	–	–	–	–	–	–
0	–	–	–	8	–	42	–	50	–	–	8	–	–	–	–	–	0	0	–	–	–	–	–
69	79	79	–	–	–	56	–	52	–	–	–	0	–	–	–	–	–	0	0	–	–	–	–
65	–	–	–	38	57	–	–	72	–	27	–	–	–	–	–	–	–	–	0	0	–	–	–
64	–	–	–	14	52	–	–	30	–	–	32	–	–	–	–	–	–	–	–	0	0	–	–
–	45	–	70	0	–	–	–	77	9	–	–	–	–	–	–	–	–	–	–	–	0	0	–
2	56	–	57	35	–	–	–	–	–	12	–	–	–	–	–	–	–	–	–	–	–	0	0
24	–	61	–	60	–	–	27	51	–	–	16	1	–	–	–	–	–	–	–	–	–	–	0

Table 3 Parity check matrix for codeword block length n=1296 bits, sub-block size is Z=54 bits

Code rate R = 1/2																							
40	–	–	–	22	–	49	23	43	–	–	–	1	0	–	–	–	–	–	–	–	–	–	–
50	1	–	–	48	35	–	–	13	–	30	–	–	0	0	–	–	–	–	–	–	–	–	–
39	50	–	–	4	–	2	–	–	–	–	49	–	–	0	0	–	–	–	–	–	–	–	–
33	–	–	38	37	–	–	4	1	–	–	–	–	–	–	0	0	–	–	–	–	–	–	–
45	–	–	–	0	22	–	–	20	42	–	–	–	–	–	–	0	0	–	–	–	–	–	–
51	–	–	48	35	–	–	–	44	–	18	–	–	–	–	–	–	0	0	–	–	–	–	–
47	11	–	–	–	17	–	–	51	–	–	–	0	–	–	–	–	–	0	0	–	–	–	–
5	–	25	–	6	–	45	–	13	40	–	–	–	–	–	–	–	–	–	0	0	–	–	–
33	–	–	34	24	–	–	–	23	–	–	46	–	–	–	–	–	–	–	–	0	0	–	–
1	–	27	–	1	–	–	–	38	–	44	–	–	–	–	–	–	–	–	–	–	0	0	–
–	18	–	–	23	–	–	8	0	35	–	–	–	–	–	–	–	–	–	–	–	–	0	0
49	–	17	–	30	–	–	–	34	–	–	19	1	–	–	–	–	–	–	–	–	–	–	0
Code rate R = 2/3																							
39	31	22	43	–	40	4	–	11	–	–	50	–	–	–	6	1	0	–	–	–	–	–	–
25	52	41	2	6	–	14	–	34	–	–	–	24	–	37	–	–	0	0	–	–	–	–	–
43	31	29	0	21	–	28	–	–	2	–	–	7	–	17	–	–	–	0	0	–	–	–	–
20	33	48	–	4	13	–	26	–	–	22	–	–	46	42	–	–	–	–	0	0	–	–	–
45	7	18	51	12	25	–	–	–	50	–	–	5	–	–	–	0	–	–	–	0	0	–	–
35	40	32	16	5	–	–	18	–	–	43	51	–	32	–	–	–	–	–	–	–	0	0	–
9	24	13	22	28	–	–	37	–	–	25	–	–	52	–	13	–	–	–	–	–	–	0	0
32	22	4	21	16	–	–	–	27	28	–	38	–	–	–	8	1	–	–	–	–	–	–	0

Appendix : Min-Sum Algorithm

- The sum-product algorithm can be modified to reduce the implementation complexity of the decoder. This can be done by altering the equation

$$R_{ji} = -2 \tanh^{-1} \prod_{i' \varepsilon Vj \backslash i} \tanh (Q_{i'j} / 2)$$

in such a way as to replace the product term by sum.

For simplicity ‘ we will write

$$\prod_{i'} \equiv \prod_{i' \varepsilon Vj \backslash i}$$

in the following presentations.

First , $Q_{i'j}$ can be factored as follows :

$$M_{i'j} = \alpha_{i'j}\beta_{i'j}$$

where $\alpha_{i'j} = sign\ Q_{i'j}$, $\beta_{i'j} = | Q_{i'j} |$

Thus we have that

$$\prod_{i'} \tanh (Q_{i'j} / 2) = \prod_{n'} \alpha_{i'j} \prod_{i'} \tanh (\beta_{i'j} / 2)$$

and then

$$R_{ji} = - 2 \tanh^{-1} (\prod_{i'} \alpha_{i'j} \prod_{i'} \tanh (\beta_{i'j} / 2)$$

$$=- 2 (\prod_{i'} \alpha_{mn'}) \tanh^{-1} \prod_{i'} \tanh (\beta_{mn'} / 2)$$

$$= - 2 (\prod_{i'} \alpha_{mn'}) \tanh^{-1} (\log^{-1} \log)\prod_{i'} \tanh (\beta_{mn'} / 2)$$

Next , we define

$$\psi(x) = -\ln \tanh(x/2) = \ln[(e^x+1)/ (e^x-1)]$$

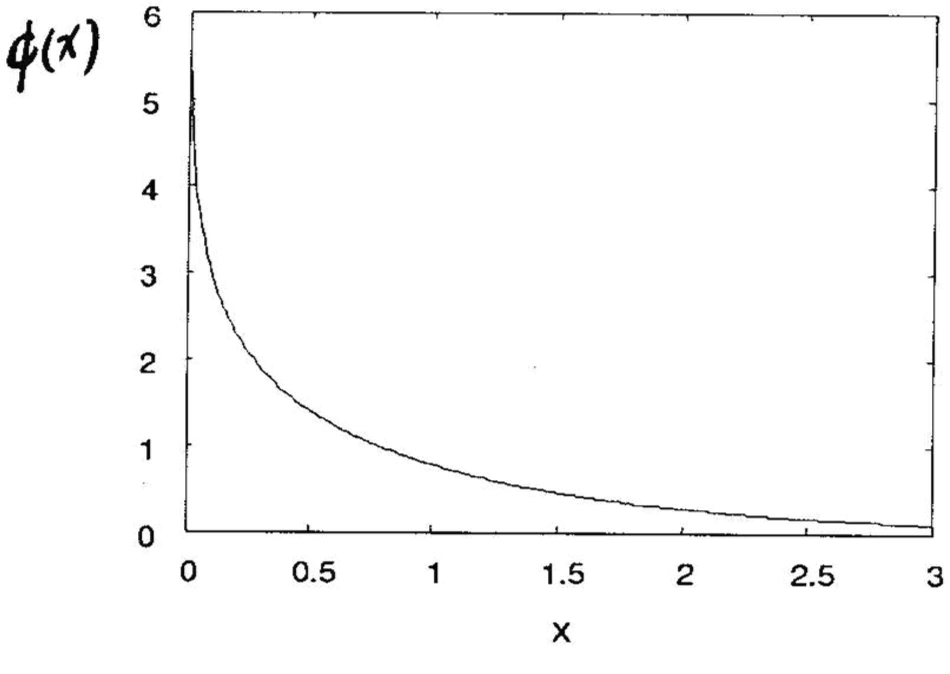
and note that $\psi (\psi(x)) = \ln[(e^{\psi(x)}+1)/ (e^{\psi(x)} -1)] = x$

$$\psi^{-1} (x) = \psi(x) \quad \text{for} \quad x> 0$$

Finally, we obtain

$$R_{ji}= - (\prod \alpha_{i'j}) \psi(\Sigma_{i'} \psi(\beta_{i'j}))$$

- **The product of the signs can be calculated by using modulo-2 addition of the hard decisions on each $Q_{i'j}$, while the function ψ can be implemented easily using a lookup table.**
- **Since the term corresponding to the smallest $Q_{i'j}$, dominates the product term and so the product can be approximated by a minimum :** $R_{ji} = -\left(\prod_{i'} \mathrm{sign}\, Q_{i'j}\right) \min_{i'} |Q_{i'j}|$


$\phi(x)$
x