Low Density Parity Check Codes

A Brief Introduction

Justin Ethier, Brandon Whalen, Nalini Devarapalli
{jethie1, bwhale1, ndevar1} @ towson.edu

Dr. Marius Zimand
COSC 683: Security and Internet Algorithms
Department of Computer & Information Sciences
Towson University, Suite 406
Towson, Maryland 21252

Abstract

The reliable transmission of data over noisy channels using high-speed encoders and decoders is a problem central to modern communication. This paper discusses the use of Low Density Parity Check (LDPC) codes to solve this problem. This paper introduces the reader to error correcting codes and provides and in-depth look at LDPC codes. We then proceed to study how Hard Decision and Probabilistic Decision Decoders work on Binary Symmetric Channels (BSC) with random noise. Simulations involving random messages sent over simulated channels are presented, along with a detailed review of each algorithm's capacity to correctly decode signals as the channel noise is increased. The original expectations were that the Hard Decision Decoder would decrease at a rate far worse when compared with the Probabilistic Decoder. Results show that as the noise in the channel increased both algorithms linearly decrease in their ability to correctly decode the signal, with the Hard Decision Decoder decreasing at twice the rate of the Probabilistic Decoder. Neither decoder performed particularly well, with the Hard Decision decoder decoding 52% of the codewords and the Probabilistic Decoder correctly decoding 75% of the with 10% of the codewords corrupted by the channel. This shows that while these implementations of LDPC are fast, they are not very useful in channels with any real noise is them.

1 Introduction

1.1 Background

In this paper we shall provide a brief introduction to Low Density Parity Check (LDPC) codes, a linear error correcting code that has gained prominence in recent years. We discuss the basic components of LDPC codes, including the construction of codes, encoding algorithms, decoding algorithms, performance metrics, real world applications, and current areas of research into LDPC codes. We shall also present an implementation of a Hard-decision decoder for LDPC codes and compare its performance to that of existing decoding algorithms.

Although a basic knowledge of communications theory is helpful, this paper does not assume that the read has any no prior knowledge of Error Correcting Codes or LDPC Codes.

1.2 Organization

Our paper is organized in the following manner. Section 2 provides a general background on Error Correcting Codes and serves as a brief introduction to the topic. Section 3 presents LDPC codes and serves as an overview of the topic. Sections 4 and 5 discuss encoding and decoding of LDPC codes. Section 6 discusses performance considerations and metrics. Section 7 discusses our semester project within the framework mapped out by the previous sections. Section 8 lists several real world applications of LDPC codes. Section 9 discusses the future directions of LDPC codes including what to expect in the next few years. Finally, Section 10 summarizes our discussion.

2 Error Correcting Codes

A central problem in communication theory is the transmission of data over a noisy channel. In his groundbreaking paper on the limits of reliable data transmission over unreliable channels, Shannon introduced the concept of codes that could be used to transmit data over such a channel. These error-correcting codes are thus the key means why which reliable transmission may be achieved.

During transmission over an unreliable channel, each bit b has a certain probability of being corrupted during transmission. Corruption is channel-dependant and may result in the bit being flipped, erased, or otherwise destroyed. In order to ensure that the recipient correctly receives a message transmitted over such a channel, it is clear that single bits cannot be transmitted. Thus, codewords corresponding to vectors over the input alphabet must be used to transmit messages. That is, a message will be encoded into a codeword prior to transmission.

A description of the properties of codewords is in order. It is assumed that all codewords have the same length, n. This length is also referred to as the block length. Each of these codewords will then in turn be used to encode a message of length $k < n$. The rate of the code may be calculated as k / n. This value is important as this is the actual percentage of transmitted bits that contain the original data; the rest of the transmission is overhead. Obviously the rate should be as close to 1 as possible to reduce the amount of transmission overhead and thus increase the speed of transmission.

A receiver may decode a message by inferring which codeword was sent. Conceptually, the receiver will do this by finding the “closest” codeword to each received vector.

Ideally codewords should be uniformly spaced with a large distance between them, so that the receiver is able to correct large numbers of transmission errors.

Finally, Shannon proved that there is a maximum capacity of a channel such that reliable transmission rates can approach this limit arbitrarily, but cannot exceed the capacity. Unfortunately Shannon's theorem did not give any clues as to how to construct, encode, or decode such codes.

3 LDPC Codes

LDPC codes are a subset of linear codes that were originally invented by Robert Gallager in the early 1960's in his PhD thesis. However at the time they were not considered practical and were generally forgotten until their recent rediscovery.

An LDPC code is a linear code that can be represented by a sparse bipartite graph. Bipartite graphs consist of 2 sets of nodes. Edges may only be drawn between nodes in different sets; thus an edge may not connect two nodes from the same set. The graph of an LDPC code, G, consists of *n* message (or variable) nodes and *r* parity check nodes. The codewords are those vectors satisfying the requirement that all message nodes connected to each check node sum to zero, as illustrated in the figure below. As we shall see later, sparsity is the key component of LDPC codes. Note that the following example is not sparse – an actual LDPC code would have far fewer edges.

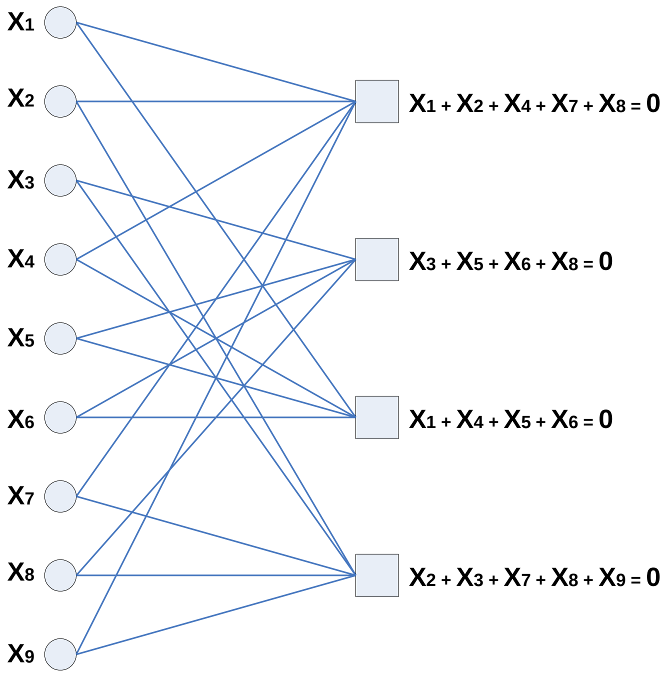

X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_8
X_9
$X_1 + X_2 + X_4 + X_7 + X_8 = 0$
$X_3 + X_5 + X_6 + X_8 = 0$
$X_1 + X_4 + X_5 + X_6 = 0$
$X_2 + X_3 + X_7 + X_8 + X_9 = 0$

Figure 1: LPDC Code Represented by a Bipartite Graph

A matrix H may also be used to represent an LDPC code. In order to transform a bipartite graph into a matrix, a matrix of size (r, n) is constructed such that an entry (i, j) is set to 1 if a message node *i* is connected to a check node *j*, and 0 otherwise. For example, the following matrix represents the same LDPC code as the bipartite graph in Figure 1.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 2: Matrix Representation of an LDPC Code

Not only does this matrix represent the code, it also corresponds to the parity check matrix of the code. The parity check matrix is significant in that it defines the null space of the code. More precisely, each codeword *w* multiplied by the transpose of H equals zero: $w \cdot H^T = 0$. Thus the parity check matrix may be used to verify that a vector is in fact a codeword.

The previous representations of an LDPC code are not truly sparse. Although every linear code may be represented by a bipartite graph, only a subset of codes may be represented by a sparse graph – these codes are considered LDPC codes. A graph is sparse if the weight of its rows and columns are significantly smaller than the size of the rows and columns. Or more formally, for a matrix of size (m, n), $w_c << n$ and $w_r << m$.

As will be shown later in the paper, the time complexity of the LDPC algorithms is tied directly to the number of edges in the bipartite graph. The fewer edges in the graph, the lower the time complexity. This is the reason why sparse parity-check matrices are desirable.

Several different algorithms exist for the construction of LDPC codes, although we will not go into their details in this paper. In fact, it is interesting that randomly chosen codes are good with a high probability [8]– however such codes will pose problems during encoding, so in practice a code must be chosen carefully.

4 Encoding

An encoding algorithm is used to transform each message from a series of bits into a codeword. For a linear code of message length k and code length n, encoding typically consists of multiplying a message m by a basis vector (g_1, g_2, … g_k). This basis vector may also be referred to as generating matrix G. The generating matrix may be obtained from the parity check matrix H as follows. First, rearrange H by Gaussian elimination so that H = [$\mathbf{P}^T$ $\mathbf{I_r}$], where I is the (r x r) identity matrix (where r is the number of check nodes). The generating matrix G may then be defined as G = [$\mathbf{I_k}$ $\mathbf{P}$]. Multiplying a message by G produces a codeword c, as noted in the equation below.

$$m \cdot G = c$$

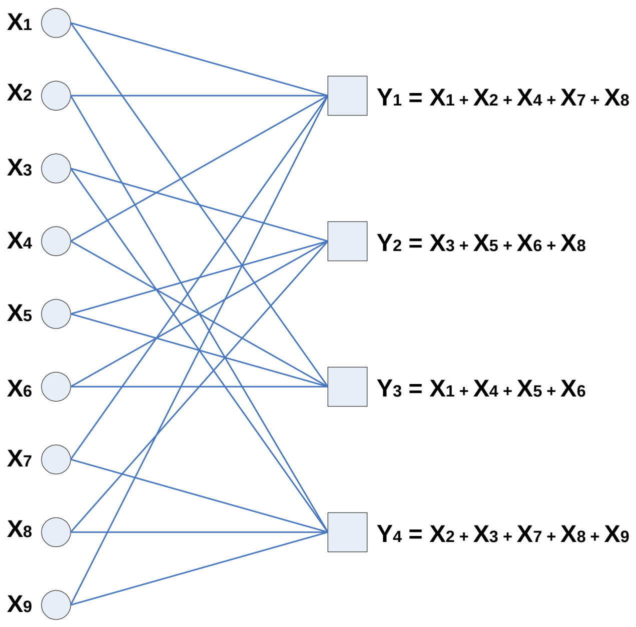
Equation 1: Standard Encoding Algorithm for a Linear Code

In this encoding algorithm, the number of operations will be dependant upon the hamming weight of the basis vectors – in other words, the number of 1's contained within the basis vectors. If the basis vectors are dense – that is, contain a large number of 1's – then the number of operations will approach *nk*. This cost is proportional to n^2 [1]. This cost may be too high for some applications – ideally, we would like a linear encoding cost of *n*.

There are several encoding algorithms documented in the literature that achieve a linear running time.

One idea to achieve linear time encoding is to store the sum of the incoming message nodes at each check node instead of using the nodes as parity constraints. In this approach the message nodes contain the original data values and are transmitted along with the check node values. It is clear that these calculations can be completed in linear time since the number of operations will only be a small constant factor more than the number of message nodes.

However there are problems with this approach. Most fundamentally the value of a check node may be corrupted during transmission. Solutions to this problem exist, such as using a layered set of graphs. That is, create a second set of redundant nodes to store information for the check nodes. This process may be continued several times, up to a logarithmic level of nodes. This can be proven to ensure reliable transmission however these workarounds increase the complexity of decoding and may cause inefficiencies in the implementation [1].



X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_8
X_9
$Y_1 = X_1 + X_2 + X_4 + X_7 + X_8$
$Y_2 = X_3 + X_5 + X_6 + X_8$
$Y_3 = X_1 + X_4 + X_5 + X_6$
$Y_4 = X_2 + X_3 + X_7 + X_8 + X_9$

Figure 3: Construction of an LDPC Code for Fast Encoding

Repeat-Accumulate (RA) codes are another set of linear codes that offer fast encoding. Their construction is similar to that of the method described above, although instead of storing sums at the check nodes a new set of nodes is created. Assuming the values of the check nodes are c_1, c_2, … c_n, these redundant nodes store the values as follows. $s_1 = y_1$, s2 $= s_1 + y_2$, etc. Thus the values accumulate as each subsequent node is calculated. We will not go into further detail on RA codes in this paper.

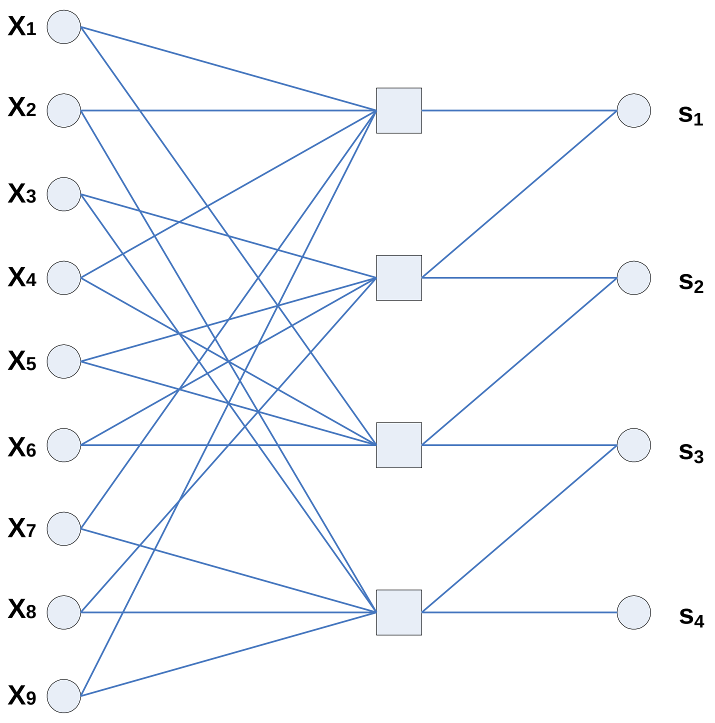


Figure 4: A Repeat Accumulate Code

5 Decoding

5.1 Overview

Decoding involves computing the most likely codeword that was sent, given the received message. A naïve approach is to compare the received message to each of the codewords, and to take the codeword that is “closest” to this message. In other words, determine the conditional probability of each codeword given the message received, and take the codeword with the highest probability. This method is referred to as Maximum Likelihood Decoding.

There are actually many advantages to Maximum Likelihood Decoding, the greatest of which is that it minimizes the probability of a decoding error. Unfortunately this method quickly becomes impractical for larger block lengths, as the number of possible codewords – and thus the running time – increases exponentially with block length. Thus, a faster decoder must be used even though the probability of a decoding error will be somewhat higher. However this is not a significant issue because the block length of a code can always be increased to reduce the probability of error [5].

The general class of decoding algorithms used for LDPC codes are called message-passing algorithms [1]. The basic idea is that each variable node sends a message to its check nodes based upon the data that it received. The check nodes in turn send a message back to each of their variable nodes based upon the values sent by the other variable nodes (neighbors) that passed a message to the check node. The variable nodes then use these values to determine if their value is correct or if it should be changed. This process continues until there are either no changes to any of the variable nodes (the message has been decoded) or a maximum number of iterations have been reached (the message could not be decoded).

We shall examine 2 variations on this idea – the passing of probabilities and the passing of binary data.

5.2 Belief Propagation

The Belief Propagation algorithm is a message-passing algorithm that passes probabilities (or "beliefs") as messages between nodes. It is worth noting that this algorithm has applications in domains other than LDPC codes, including Artificial Intelligence.

At each iteration of the belief propagation algorithm, a variable node passes messages to each of its check nodes containing the probability that the variable node has a certain value. Each check node then sends a message back to its variable nodes containing the probability that the variable node has a certain value based upon the probabilities passed by the other variable nodes to that check node.

A binary random variable x an equation L(x) may then be defined as the likelihood of x, as shown below. Note that in practice for actual computations log likelihoods are more convenient to use than likelihoods, so the following equations in this section use log-likelihoods.

$$L(x) = \frac{\Pr[x=0]}{\Pr[x=1]}$$

Equation 2: Likelihood of x

Since we are using message passing between the nodes, we would like to define a second random variable, y, such that the probability of x will be conditional upon y. We can then calculate the conditional probability of x given y1, y2, … yn as the sum of all conditional probabilities ln L(x | yi). Assuming statistical independence of digits, we can extend this to multiple x digits by observing that the probability of two independent events is equal to the product of their probabilities. Thus we can extend this to x1, x2, … xn as follows: $2\Pr[x1 \oplus x2 \oplus ... x\ell \mid y1, y2, ... y\ell) - 1 = \prod_{i=1}^{\ell} (2\Pr[xi = 0 \mid yi] - 1$. We can transform this into an equation for likelihoods using $2\Pr[xi = 0 \mid yi] - 1 = \frac{L-1}{L+1} = \tanh(\frac{\ln L}{2})$. Thus we obtain the following equation for calculating the log-likelihood that a received digit is a certain value.

$$\ln L(x1 \oplus x2 \oplus ... x\ell \mid y1, y2, ... y\ell) = \ln \frac{1 + \coprod_{i=1}^{\ell} \tanh(\frac{\ln L}{2})}{1 - \coprod_{i=1}^{\ell} \tanh(\frac{\ln L}{2})}$$

Equation 3: Equation for the Log-Likelihood of a Received Digit

Given equation 3, the following equations may be derived to formally describe the messages passed by the algorithm at each step.

$$m_{vc}^{(\ell)} = \begin{cases} m_v, & if\,(\ell = 0), \\ m_v + \sum_{c' \in C_v \setminus \{c\}} m_{c'v}^{(\ell-1)}, & if\,(\ell \geq 1) \end{cases}$$

Equation 4: Equation for Messages sent to Check Node under Belief Propagation

$$m_{cv}^{(\ell)} = \ln \frac{1 + \prod_{v' \in V_c \setminus \{v\}} \tanh(\frac{m_{v'c}^{(\ell)}}{2})}{1 - \prod_{v' \in V_c \setminus \{v\}} \tanh(\frac{m_{v'c}^{(\ell)}}{2})}$$

Equation 5: Equation for Messages sent to Variable Nodes under Belief Propagation

Since Equation 3 assumes that the probabilities for each digit are statistically independent of the other digits, these message passing equations are really only valid for the first few iterations where this independence holds. However, in practice this requirement is often ignored on the assumption that these dependencies have only a minor effect on the outcome [5].

5.3 *Hard-Decision Decoding*

Although belief propagation is the best algorithm among message passing algorithms [1], it is overly complex. Hard-Decision decoding is a simplified version of the belief propagation algorithm that allows for an easier, more efficient implementation. Under hard-decision decoding, discrete binary data – that is, a 0 or 1 – is sent in the messages instead of probabilities. Binary data is used since it is the simplest possible form of data that may be passed.

The steps of the hard decision algorithm are as follows:

1. The message nodes each contain a value 0 or 1 that was received from the channel. These values are then used as the initial messages sent to the neighboring check nodes.

2. Each check node then uses the messages received from each of its variable nodes to compute a response. The response is calculated as the sum (mod 2) of the values received from all variable nodes except the one that each message will be sent to.

   The algorithm may also terminate at this step if each summation is zero, meaning that all the parity-check equations are met. Optionally, the algorithm may also terminate if it has run for a predefined number of iterations – for example, 200.

3. Each variable node uses the messages received from its check nodes to determine if its original value is correct. There are different ways to make this determination depending upon which version of the algorithm is used:

   a. The Gallager A algorithm decides to use the received value, v, if v is the same for all messages received from the check nodes. Otherwise the node retains its value from the previous iteration.

   b. The Gallager B algorithm improves upon A and uses a threshold value $b_{i,j}$ such that the value is only changed if at least $b_{i,j}$ neighbors sent the same value in the previous round. In his original version of the algorithm, Gallager set b to be $j/2$ for j even or $(j + 1)/2$ for j odd, where j is the number of messages received from the check nodes. Alternatively this can be seen as a "majority vote" of the received values. Note that Gallager A is really just a special case of B in which $b_{i,j} = j - 1$ independent of the round [1].

4. Go back to step 2.

We can now see why a sparse parity check matrix is advantageous. The number of messages passed in each iteration will largely determine the running time of this algorithm. With a sparse matrix, the number of messages is minimized, thus reducing the running time.

5.4 Example of Hard-Decision Decoding

Let us now illustrate how the hard decision algorithm works with a simple example. We will use an LDPC code based on the (4, 8) bi-regular graph shown below. This size graph is actually the best-performing graph for the Gallager A algorithm, although we do not guarantee the performance of the graph we present.

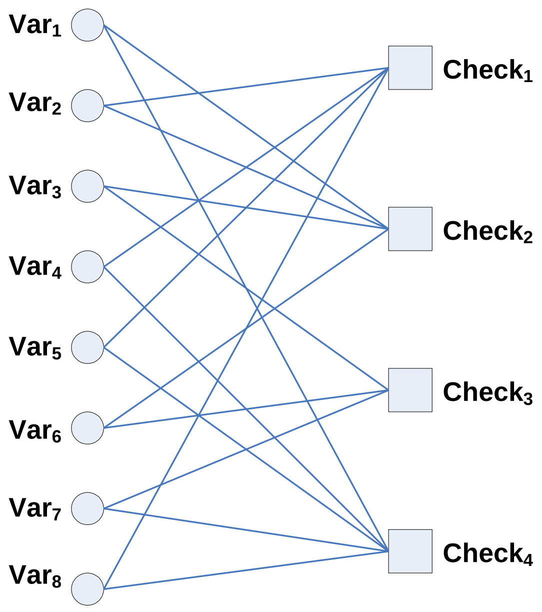


Figure 5: Bipartite (4, 8) Graph

A valid codeword for this code is `10101001`. Now let us assume that a message transmitted using this codeword was received as `00101001` with a 1-bit error in the first position. The hard-decision algorithm should correct this error during decoding. The following tables illustrate the messages passed during the first iteration of the algorithm.

Check Node	**Variable Node**	**Message Sent to Check Node** V → C	**Response** C → V
1	2	0	0
1	4	0	0
1	5	1	1
1	8	1	1
2	1	0	1
2	2	0	1
2	3	1	0
2	6	0	1
3	3	1	1
3	6	0	0
3	7	0	0
3	8	1	1

4	1	0	1
4	4	0	1
4	6	1	0
4	7	0	1

Table 1: Messages passed from Variable Nodes to Check Nodes at First Iteration

Variable Node	**Original Value**	**Responses** C → V	**Value After First Iteration**
1	0	{1, 1}	**1**
2	0	{0, 1}	**0**
3	1	{0, 1}	**1**
4	0	{0, 1}	**0**
5	1	{1, 0}	**1**
6	0	{0, 1}	**0**
7	0	{0, 1}	**0**
8	1	{1, 1}	**1**

Table 2: Messages passed from Check Nodes to Variable Nodes at First Iteration

Table 1 illustrates the messages passed from variable nodes to check nodes along with the calculated responses, and illustrates the messages received by the variable nodes along with the final variable node values after this iteration. As you can see, after this iteration the bit error is corrected and the codeword has been successfully decoded. During iteration 2, the algorithm will calculate the parity at each check node to be 0 and will terminate, returning the correct codeword that may then be converted into the original message.

We further discuss and present experimental results of this algorithm in our project section.

6 Performance

Besides running times, which we have already discussed, the other performance metrics to consider are errors rates. There are 2 main types of errors – bit errors and block errors. From a practical point of view, block errors are more important as they imply that source data could not be successfully decoded. Bit error rates are more often cited in the literature, perhaps due to historical reasons [6].

Block error rates decrease towards 0 as the block length of the code increases towards infinity. Thus increasing the length of the code will, in general, decrease the error rate of a code.

One excellent property of LDPC codes is that the algorithm reports any decoding errors. Decoding algorithms for other types of codes such as Reed Solomon or Turbo codes simply iterate a certain amount of times and return a value – there is no way to know the accuracy of the results.

7 Project

For our project we implemented a hard decision decoder based upon the Gallager algorithm, with slight modifications. In order to expedite our implementation we selected an existing LDPC code software package to build our program on top of. After some research we selected an excellent set of programs written in C by Radford M. Neal [8]. As can be seen in the diagram below, this software contains a full suite of applications for constructing, encoding, transmitting, decoding, extracting, and verifying LDPC codes.

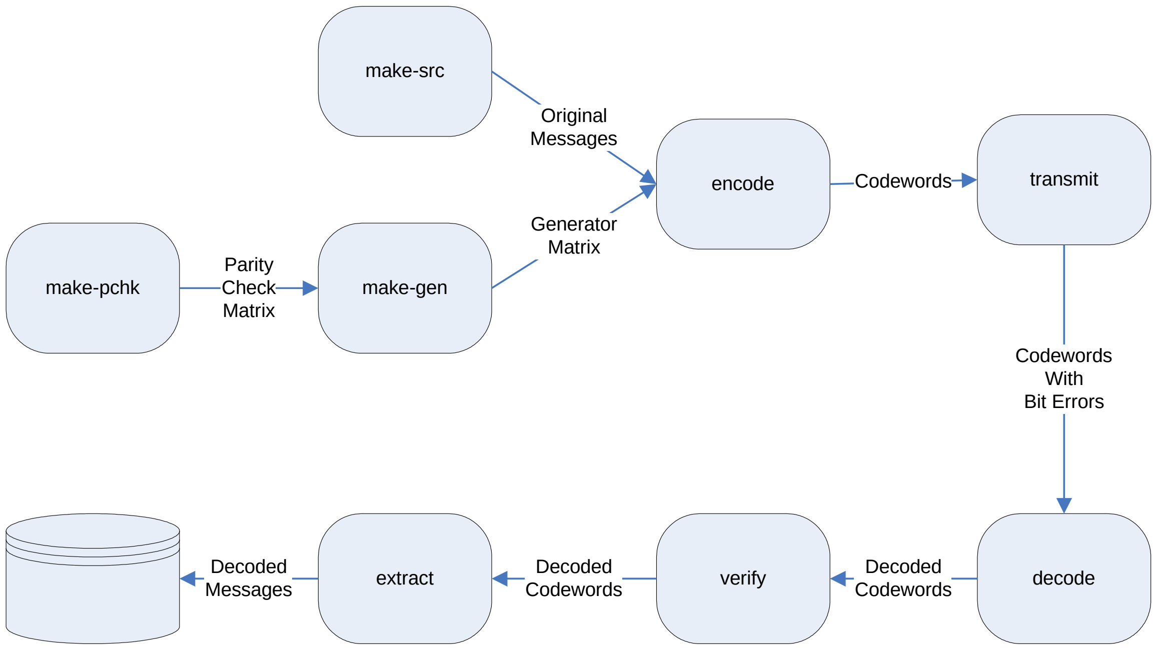


Figure 6: Overview of Radford M. Neal's LDPC Code Software Package

Neal's software package serves as the framework for our project. It provides support for a variety of communication channels, including the Binary Symmetric Channel (BSC) that we used for our experiments. In addition the suite provides implementations of several algorithms including the Gallager Probability Propagation algorithm, which we have used as a baseline for evaluating our algorithm.

The existing framework required us to add two functions to dec.c, one to initialize any data structures, and a second to do the decoding. We were also required to add defines into the existing headers so that the user could call our decoding functions, and finally had to edit decode.c so that it could call our new functions.

Our decoder is built using two structures that represent the hard decoding algorithm. The message nodes, also called v nodes, are represented by a linked list of v_node_t. Each has a value that represents the current value of that node, and an index, which is that nodes index in the message. The check nodes are represented an array of c_node_t structures that each has 2 linked lists, one for the received messages and one for the sent messages. This design decision was made so that we could calculate the sent and received messages at each round.

The algorithm our decoder uses works as such:

```
Initialize the hard decode structures
Check to see if we have solved the array, i.e. did this codeword come in correct
While (we have not solved the array and we haven’t exceeded our maximum iterations)
        Calculate what was sent to the check node
        Calculate what the message nodes received from the check nodes.
        Recalculate the answer array
Done while
Return the number of iterations to complete
```

It is important to note that solving the array is performed by multiplying the parity check matrix by the answer array. If all of the resulting bits are 0 the answer is said to be found. As the codeword gets more corrupted the algorithm will generate an incorrect answer array that still solves this equation.

The experiments were conducted by creating a 4 x 8 sparse matrix to do encoding and decoding. Generating a random message of 4000 bytes, and then encoding that message using the generating matrix. The message was then ran through a Binary Symetric (BSC) channel with an error probability varying from .001 to .01 increasing at each step by .001. Finally the message was decoded using our hard decoding algorithm with a maximum of 100 possible iterations. In order to provide a baseline, we also ran the same data from the channel through an implementation of the of Gallaghers Probability Propogation algorithm. All experiments and coding were conducting under the Fedora Core Linux distribution.

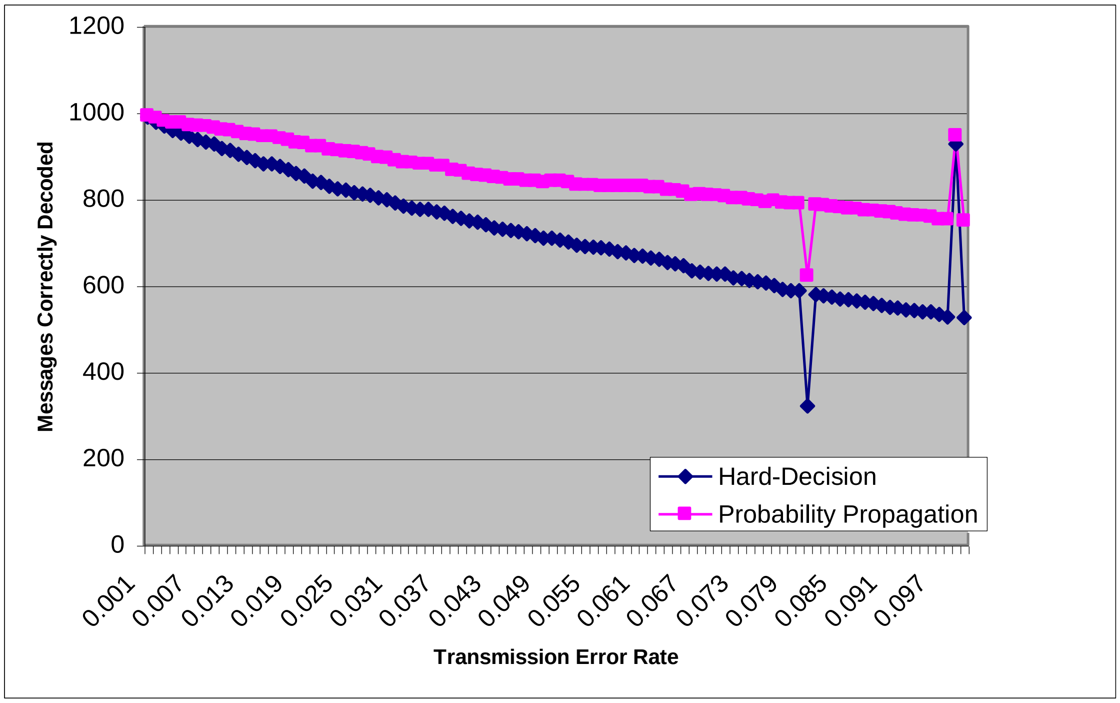


Figure 7: Error Rates for LDPC Decoders

Our original thoughts on the decoders were that our hard decision decoder would perform far worse than the probabilistic decoder because it was so simplistic. As you can see from the graph the probability decoder and the hard decision decoder both perform linearly with the error rate in the codeword. The Probability decoder decreases at a rate of 2.3 correct for every .001 increase in error, while the Hard decoder decreases at a rate of 4.3 correct for the same increase. This shows that the hard decision decoder works about half was well as the error increases. This is actually better than we thought it would perform, and shows that while the Hard decoder is very simple it still is able to achieve a relatively good decoding rate in comparison with the Probabilistic version.

8 Real World Applications

For real world communication engineers the Shannon limit can be seen as virtuous especially those in the networking and wireless fields. Since being first defined in the late 1940s, designers have developed and implemented error correction coding techniques to push channel performance closer and closer to the Shannon limit. For the past few years, turbo coding was considered as the key technique for improving channel performance. But LDPC codes have achieved a recent resurgence and are now considered the some of the most promising codes to bring designers closer to the Shannon Limit.

The advanced communication and the coding techniques offered by LDPC codes could allow the operation of DSL (Digital Subscriber Lines) links much closer to their confined abilities than is the case with current state-of-art systems. The anticipation of LDPC codes meets necessities of ADSL and VDSL transmissions. LDPC codes have several applications in this domain, including:

- LDPC codes have mainly been considered for the data transmission systems employing binary modulation.
- The collection of all the technologies that are able to transmit data at high speeds over standard twisted pair lines that currently come in to homes is DSL. LDPCs however have recently begun to gain attention because they are parallelizable, and VLSI technology has gotten to the point where LDPCs over fairly large code word sizes can be implemented in hardware.
- LDPC codes are decoded in iterative fashion by the computationally simple sum-product algorithm at the expenses of increased storage requirements [1].
- LDPC codes have the ability of achieving tremendous accomplishment with out showing the signs of “error floors”.
- LDPC codes acknowledge the trade-offs between performance and decoding complexity.

The integral part of the design of modern communications systems takes advantage of LDPC coding techniques that provide considerable growth in performance. It also allows an increase in data rate that can be influential to the widespread deployment of future DSL service.

9 Future

LDPC is attracting so much attention from so many different sectors because it offers significant effects in the form of faster communications, longer communication ranges and better transmission quality. For example:

- Well known wireless Internet Protocol developed by the Flarion technologies used the LDPC code processing.
- LDPC coding has been adopted by the IEEE802.3an Task Force, for working on the 10Gbit/s Ethernet over twisted pair.
- The DVB-S2 standard for satellite digital broadcasting with high-definition TV (HDTV) capability also adopted it as a standard in the final draft, which was released June 2004 [2].
- Due to its longer range and higher capacity, organizations like IEEE802.3an and DVB-S2 implemented the scheme. It makes it possible to pump data over twisted pair at 10Gbit/s Ethernet, to distances of up to 100m, which was formerly thought impossible [2]. At the same time, it also increases the transmission capacity of a single satellite broadcast TV transponder in the 36MHz waveband to 80Mbit/s, representing a 1.3x improvement [2].

Other applications for which LDPC is currently being considered are Ultra-high speed wireless local area networks (LAN), Optical communications, HDD (Hard Disk Drives) signal processing circuits, Quantum-encrypted communication, and many more.

Researches are investigating the use of advanced error correction coding for digital audio and video broadcasting, as well as for increasing data rates in wi-fi networks, wireless LAN. LDPC codes are proving to give excellent coding gains over a wide range of code rates and block sizes.

Liaison Statement To CCSDS Management Council states that LDPC codes offer significant advantages to future missions operating at very high data rates [3]. Their potential for significant coding gain with only a modest increase in bandwidth makes them very attractive. Several agencies are planning ahead to make use of the LDPC code on their future high data rate space missions.

LDPC codes are also capable of exceptional performance on channels where data is not only tarnished but may be lost entirely, so-called erasure channels. This initiated the new application domains such as reliable Internet multicasting where whole packets of lost data are reconstructed without the network overhead of retransmission. The challenge, then, is to devise new LDPC codes with sufficient flexibility to cope with the numerous applications opening up in future generation wireless communications, data storage and the Internet.

10 Conclusion

a. This paper has presented LDPC codes, a newly revitalized class of error correcting codes that can help engineers move closer to the Shannon Limit. The encoding and decoding can be done quickly and efficiently due to the fact that the algorithm uses a sparse matrix and only few entries need to be calculated. An example simulation was presented using a simplistic decoder allowing the user to see how the class of decoders works on noisy channels. More simulations could be done to see how the Hard Decision decoder works on channels with other kinds of noise, and how varying the generating matrix affects the effieciency and correctness of the decoder. As more work is done is the field of error correcting codes it will only increase our ability to quickly and reliably transmit data as more and more of the world uses digital communications.

References

1. LDPC Codes: An Introduction, Amin Shokrollah, http://www.ics.uci.edu/~welling/teaching/ICS279/LPCD.pdf, Digital Fountain, Inc., 39141 Civic Center Drive, Fremont, CA 94538, April 2, 2003.

2. On Advanced Signal Processing and Coding Techniques for Digital Subscriber Lines, Giovanni Cherubini, Evangelos Eleftheriou, and Sedat Ölçer, IBM Zurich Research Laboratory.

3. LDPC Adopted for Use in Comms, Broadcasting, HDDs, http://neasia.nikkeibp.com/neasia/00828, NEAsia April 2005 Issue.

4. Liaison Statement To CCSDS Management Council, Liaison Statement 2

5. Low-Density Parity-Check Codes, Robert G. Gallager, http://www.inference.phy.cam.ac.uk/mackay/gallager/papers/, M.I.T. Press, Cambridge, MA, 1963.

6. LDPC Codes, Jeremy Thorpe, http://www.ldpc-codes.com/index.html, Caltech, May 2006.

7. LDPC Codes – A Brief Tutorial, Bernhard M.J. Leiner, http://geilenkotten.homeunix.org/ldpc.pdf, April 8, 2005.

8. Software for Low Density Parity Check Codes, Radford M. Neal, Dept of Statistics and Dept. of Computer Science, University of Toronto, http://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/index.html, May 2006.