DISTRIBUTED SOURCE CODING USING NON BINARY LDPC

A Thesis by

Madhurima Potluri

Bachelor of Engineering, Visweswaraiah Technological University, India, 2002

Submitted to the College of Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2005
DISTRIBUTED SOURCE CODING USING NON BINARY LDPC

I have examined the final copy of this Thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

__________________________________
Dr. Kamesh Namuduri, Committee Chair

We have read this Thesis and recommend its acceptance:

__________________________________
Dr. James E Steck, Committee Member

__________________________________
Dr. Larry Paarman, Committee Member
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Kamesh Namuduri, for his guidance and support. I would like to extend my gratitude to the members of my committee, Dr. James E Steck, and Dr. Larry Paarman, for their helpful comments and suggestions. I also want to thank my family and friends.
ABSTRACT

In this thesis we have extended non binary low-density parity-check (LDPC) codes, developed for channel coding, to be used for source coding to compress correlated non binary sources. Focusing on the asymmetric case of compression of an equiprobable memoryless non binary source with side information at the decoder, the approach is based on viewing the correlation as a channel and applying the syndrome concept. The encoding and decoding procedures are explained in detail. The results obtained through simulations showed that the non binary compression scheme gives symbol error rates of zero for correlation coefficients greater than or equal to 0.80 for a compression rate of 1/2. For a compression rate of 3/4 the non binary compression scheme gives symbol error rates of zero for correlation coefficients greater than or equal to 0.65. In comparison the binary compression scheme gives bit error rates of zero for correlation coefficients greater than or equal to 0.90 for compression rate of 1/2. For compression rate of 3/4 the binary compression scheme gives bit error rates of zero for correlation coefficients greater than or equal to 0.75. This shows that the non binary compression scheme handles sources with relatively low correlations better than the binary scheme.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Need for Source Coding</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Thesis Overview and Contribution</td>
<td>3</td>
</tr>
<tr>
<td>2.</td>
<td>DISTRIBUTED SOURCE CODING</td>
<td>6</td>
</tr>
<tr>
<td>3.</td>
<td>LOW DENSITY PARITY CHECK CODES</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction to Linear Block Codes</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Introduction to LDPC Codes</td>
<td>10</td>
</tr>
<tr>
<td>3.3</td>
<td>Encoding and Decoding</td>
<td>12</td>
</tr>
<tr>
<td>3.4</td>
<td>Non binary LDPC Codes</td>
<td>19</td>
</tr>
<tr>
<td>4.</td>
<td>DISTRIBUTED SOURCE CODING USING BINARY LOW DENSITY PARITY CHECK CODES</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Encoding and Decoding with Binary LDPC Codes</td>
<td>24</td>
</tr>
<tr>
<td>5.</td>
<td>DISTRIBUTED SOURCE CODING USING NON BINARY LDPC</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>Encoding and Decoding with Non Binary LDPC Codes</td>
<td>29</td>
</tr>
<tr>
<td>6.</td>
<td>SIMULATION AND RESULTS</td>
<td>34</td>
</tr>
<tr>
<td>6.1</td>
<td>Distribute Source Coding using Binary LDPC</td>
<td>34</td>
</tr>
<tr>
<td>6.2</td>
<td>Distributed Source Coding using Non Binary LDPC</td>
<td>34</td>
</tr>
<tr>
<td>6.3</td>
<td>Observations and Results</td>
<td>39</td>
</tr>
<tr>
<td>7.</td>
<td>CONCLUSIONS</td>
<td>40</td>
</tr>
<tr>
<td>8.</td>
<td>BIBLIOGRAPHY</td>
<td>41</td>
</tr>
<tr>
<td>9.</td>
<td>APPENDIX</td>
<td>44</td>
</tr>
</tbody>
</table>
# TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Slepian-Wolf rate region for two sources.</td>
<td>7</td>
</tr>
<tr>
<td>Rate distortion with side information.</td>
<td>8</td>
</tr>
<tr>
<td>System for compression with side information.</td>
<td>9</td>
</tr>
<tr>
<td>Correlation channel for compression with side information.</td>
<td>9</td>
</tr>
<tr>
<td>Example of a parity check matrix for $N=20, j=3$ and $k=4$.</td>
<td>12</td>
</tr>
<tr>
<td>Message passing on the bipartite graph representing a parity-check matrix</td>
<td>13</td>
</tr>
<tr>
<td>Comparison of graph structure of non binary and binary LDPC.</td>
<td>22</td>
</tr>
<tr>
<td>Modeling Correlation using BSC.</td>
<td>23</td>
</tr>
<tr>
<td>Plot for Distributed source coding using binary LDPC.</td>
<td>34</td>
</tr>
<tr>
<td>Plot for Distributed source coding using non binary LDPC over $GF(4)$ for compression rate 1/2</td>
<td>35</td>
</tr>
<tr>
<td>Plot for Distributed source coding using non binary LDPC over $GF(8)$ for compression rate 1/2</td>
<td>36</td>
</tr>
<tr>
<td>Plot for Distributed source coding using binary and non binary LDPC over $GF(4)$ for compression rate 1/2</td>
<td>36</td>
</tr>
<tr>
<td>Plot for Distributed source coding using non binary LDPC over $GF(4)$ for compression rate 3/4</td>
<td>37</td>
</tr>
<tr>
<td>Plot for Distributed source coding using non binary LDPC over $GF(8)$ for compression rate 3/4</td>
<td>38</td>
</tr>
<tr>
<td>Plot for Distributed source coding using binary and non binary LDPC over $GF(4)$ for compression rate 3/4</td>
<td>38</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1 Need for Source Coding

Source coding seeks to minimize the bit rate of a signal without an objectionable loss of signal quality in the process. High quality is attained at low bit rates by exploiting the source characteristics, the signal redundancy as well as the knowledge that certain types of coding distortion are indiscernible because they are masked by the signal. Techniques exploiting the source characteristics, such as signal redundancy and distortion masking are becoming increasingly more sophisticated, leading to the continuing improvement of low bit rate signals. The capability of signal compression has been central to the technologies of robust long distance communications, high quality information storage and information encryption. Compression continues to be a key in communications in spite of the promise of optical transmission media of relatively unlimited bandwidth due to the continued and increasing need to use band limited media such as radio and satellite links. Achieving a low bit rate is a key factor in meeting the demands of the new digital wireless communication systems. Impressive progress has been made in this area in recent years. Distributed source coding is one such technology.

Representing the source signal by a sequence of symbols from some finite alphabet, and then coding the alphabet symbols into fixed length blocks is one of the simplest source coding techniques. Simple examples include, encoding of twenty-six English alphabets into five bit blocks or converting many special symbols into 8-bit blocks using the standard ASCII code. The basic objective of the source encoder is to encode as efficiently as possible. It has to transmit as few bits as possible, subject to the need to reconstruct the input adequately at the output. In such cases source encoding is often called data compression. Amplitude quantization, differential
pulse code modulation, adaptive prediction, block coding and transform coding are some examples of source coding. Amplitude quantization is the assignment of a number to the amplitude of a wave. In differential pulse code modulation, an analog signal is sampled and the difference between the actual value of each sample and its predicted value, derived from the previous sample or samples, is quantized and converted by encoding to a digital signal. Adaptive prediction is a time-varying process that computes an estimate of the input signal from the quantized difference signal. Block coding is a type of coding that encodes strings formed from an alphabet set $S$ into code words by encoding each letter of $S$ separately. Transform coding is used to convert spatial image pixel values to transform coefficient values. Source coding for digital data is a way of processing data in order to reduce redundancy or prepare it for later processing. In this thesis we consider the distributed source coding for digital data. A method is developed for compressing both binary and non-binary sources using low density parity check codes.

Consider a sensor network consisting of sensors that are gathering information from a common environment. These sensors send the highly correlated information to a data gathering node, which forms an amalgamated view of the environment being sensed, based on a fusion of information collected by all the sensors. As the sensors have highly correlated information, communication among them will result in removal of redundant information and will also reduce the bandwidth of the transmission channel between the source (sensors) and the destination. This can be achieved as a consequence of the information theoretic bounds established by Slepian and Wolf [1] for distributed lossless coding, and by the Wyner and Ziv [14] for lossy coding using the decoder side information. One of the enabling technologies for the implementation of these theoretic bounds is distributed source coding (DSC), which exploits the source statistics at the decoder by using a simple encoder. DSC [2] is the compression of multiple correlated sensor
outputs that do not communicate with each other, but send their compressed outputs to a common decoder for joint decoding. This results in increased complexity at the decoder, reversing the traditional complex encoder and simple decoder. Such systems are suitable for wireless sensor networks as the complexity of the transmitter at the sensor is reduced enabling the design of sensor transmitters that are less complex. Driven by a host of emerging applications like remote sensing, military applications and wireless video networks, practical schemes for the implementation of the well know Slepian-Wolf [1] information theoretic bounds have appeared recently and are being explored. Such coding schemes have many other applications and hold a great promise for the new generation wireless communications as they can be applied in reliable communications to real world problems that will prove extremely exciting and will yield fruitful results.

1.2 Thesis Overview and Contribution

This thesis first reviews the implementations of distributed source coding techniques. A new method of distributed source coding for non binary sources using low density parity check codes is developed and implemented. In addition, a new scheme for distributed source coding of binary sources using low density parity check codes is developed and implemented in the probability domain as opposed to the log domain presented by Liveris et al [2]. The performance of these schemes is analyzed by comparing the bit error rate and symbol error rate for different correlations between two randomly generated binary and non binary sources respectively.

Gallager’s [6] low density parity check (LDPC) codes are defined by sparse parity check matrices, usually with a random construction. Such codes have near Shannon limit performance when decoded using an iterative probabilistic decoding algorithm. Low density parity check codes are also shown to be useful for communicating over channels which make insertions and
deletions as well as additive (substitution) errors. LDPC codes were extended over non binary sources for channel coding by Davey et al [9]. These codes were shown to have a 0.6 db improvement in signal to noise ratio for a given bit error rate. The use of LDPC codes for distributed source coding was suggested by Liveris et al [2]. They developed distributed source coding scheme for binary sources in the log domain. This thesis deals with the development of distributed source coding for non binary sources using LDPC codes.

The first scheme is the implementation of distributed source coding scheme proposed in the paper [2]. The implementation in this thesis is carried out using LDPC codes in the probability domain as opposed to log domain described in [2]. This was necessitated as the development of distributed source coding for binary sources over a log domain could not be directly extended for non binary sources. Two binary sources with different correlations are considered. One of the sources is assumed to be available at the decoder and is acting as the side information. The other source is compressed and sent to the decoder. The decoder decodes the compressed source by using the side information available to it from the other source. The performance of this scheme is evaluated for different correlations between the two binary sources. The bit error rates are computed and plotted.

One of the aims of this thesis is transmission of video frames using the above mentioned scheme. Video frames are highly correlated which lend themselves well for transmission using distributed source coding. The above mentioned scheme is for binary domain. This necessitated the conversion of the video frames into its binary equivalent. On conversion of the video frames into binary sequences, it was observed that the correlation between the video frames decreased drastically thereby rendering the above scheme unsuitable for video frame transmission. This led to the investigation of modifying non binary LDPC for distributed source coding which would
allow the transmission of video frames without conversion to binary sequences thereby preserving the correlation. Thus the second scheme developed in this thesis provides an ideal way of implementing distributed source coding for non binary sources like video sources. The scheme devised for distributed source coding in the non binary domain is implemented for different Galois fields.

The performance of this scheme is evaluated for different correlations between the two non binary sources. The above scheme is implemented for Galois field 4 (\(GF(4)\)) and Galois field 8 (\(GF(8)\)) non binary fields. The performance evaluation of the methods developed in the thesis was done using simulations on Matlab. One of the non binary source of specified length is generated using random number generation, the other sequence is generated such that a given correlation exists with the first sequence. The second sequence is now decompressed using the first sequence. Higher order Galois fields and video frame transmissions were not simulated due to memory constraints. The performance of the above scheme is quantified by using sources with different correlations as well as different compression rates at the encoder. The symbol error rates are computed for the above cases and are plotted.
2. DISTRIBUTED SOURCE CODING

Source coding is a way to remove the naturally occurring redundancy in the input signal so as to reduce the bandwidth of the signal for it to be accommodated in the channel. Source coding methods can be classified into, lossless or lossy source coding. Lossless source coding is the compression of a signal where in the decompression gives back the original signal. Slepian Wolf coding is a case of lossless source coding. Lossy source coding on the other hand achieves greater compression by throwing away some parts of the signal that really don’t matter. Wyner-Ziv coding is an example of lossy source coding. The source statistics play an important role in successful source coding. Shannon showed that there is a limit to which a source can be compressed without introducing errors at the decoder. Rate distortion theory gives a trade off between compression and quality in lossy source coding. Images, video and audio are often compressed using lossy source coding to achieve better compression techniques, however the compressors will have a lossless mode. Moreover, a lossless algorithm can be used as a building block in designing a lossy compressor.

Distributed source coding of correlated sources, is the compression of correlated sources that send their outputs to a common decoder without communicating with each other. It has evolved out of the need for removal of redundancy of data in networks involving applications using dense sensor networks. The problem of lossless compression of finite alphabet sources takes its roots from the fundamental paper of Slepian and Wolf [1]. The Slepian Wolf theorem states that the output of two correlated sources can be compressed to the same extent without loss, whether they communicate with each other or not provided that the decompression takes place at the joint decoder. If we consider two sources $X$ and $Y$ for separate encoding of these two
sources rates $R_x > H(X)$, where $H(X) = -\sum_{x \in X} p(x) \log p(x)$ is the entropy of $X$ and $R_y > H(Y)$, $H(Y) = -\sum_{y \in Y} p(y) \log p(y)$ being the entropy of $Y$ are required. A rate $H(X,Y)$, which is the joint entropy of $X$ and $Y$ defined as $H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x,y)$ is required for joint encoding of both the sources. However, the paper by Slepian and Wolf shows that a total rate $R = H(X,Y)$ is sufficient even for separate encoding of correlated sources. The Slepian Wolf rate region for two arbitrarily correlated sources $X$ and $Y$ is as shown in the figure 2.1 and is bounded by the following inequalities.

$$R_x \geq H(X/Y), \quad R_y \geq H(Y/X), \quad R_x + R_y \geq H(X,Y)$$

![Figure 2.1 The Slepian-Wolf rate region for two sources.](image)

Wyner and Ziv have extended the Slepian Wolf theorem to continuous valued Gaussian sources. According to Wyner and Ziv for two correlated sources $X$ and $Y$ shown in figure 2.2, the rate
distortion performance obtained for encoding $X$ is the same whether the encoder has an understanding of $Y$ or not, if $Y$ is available at the decoder.

Figure 2.2 Rate distortion with side information.

The Wyner Ziv rate distortion function gives the minimum rate $R$ necessary to reconstruct the source $X$ with average distortion less than or equal to $D$.

$$R_y(D) = \min_{p(u|x), f(y,u)} I(X;U) - I(Y;U)$$

Minimization is performed over all $p(u|x)$ and $f(y,u)$ and all decoder functions satisfying fidelity constraints and $U$ is an auxiliary random variable such that $|U| \leq |X| + 1$ and $Y \rightarrow X \rightarrow U$ form a Markov chain.

Applications involving distributed source coding are data compression for network communications, sensor networks, upgrading of existing schemes and video compression. In video compression the motivation is to reduce the complexity of the encoder at the expense of increased complexity at the decoder.

The Slepian Wolf and Wyner Ziv theorems only provide theoretical asymptotic bounds for lossless and lossy distributed source coding of correlated sources. They do not include practical solutions for the design of encoders and decoders, for implementation of the source coding schemes. Most of the practical solutions for the implementation of distributed source
Coding are derived from channel coding concepts. The statistical dependence or correlation between the sources is modeled as a virtual correlation channel equivalent to a binary symmetric channels or additive white Gaussian channels (AWGN). Considering two correlated sources $X$ and $Y$ the source $Y$ is considered as the side information or noisy version of $X$, the main source as shown in figure 2.3 and figure 2.4.

![Figure 2.3 System for compression with side information.](image)

![Figure 2.4 Correlation channel for compression with side information.](image)

Practical solutions for the design of encoders for the sources are then obtained based on channel codes like block codes; convolutional codes, turbo codes and low density parity check codes.
3. LOW DENSITY PARITY CHECK CODES

3.1 Introduction to Linear Block Codes

An \((N,K)\) block code \(C\) is a mapping of a message vector of length \(K\) to an codeword \(C\) of length \(N\). The code \(C\) is linear if it is a \(K\) dimensional subspace of an \(N\) dimensional binary vector space \(V_N\), it can also be viewed as a mapping of \(K\)-space to \(N\)-space by a \(NK \times 1\) generator matrix \(G\), where \(C = mG\). The rows of \(G\) constitute a basis of the code subspace. The dual space, \(C^\perp\) consists of all those vectors in \(V_N\) orthogonal to \(C\) so for all \(c \in C\) and all \(d \in C^\perp\) then \(<c,d>=0\). The rows of the \(N-K \times N\) parity check matrix \(H\) constitute the basis for \(C^\perp\) so for each \(c \in C\), \(cH^\perp = 0\). Therefore a linear code is completely specified by either \(G\) or \(H\) matrix.

3.2 Introduction to LDPC Codes

Low density parity check codes [LDPC] are a class of linear error correcting block codes introduced by Gallager in 1962 [6] and rediscovered by Mackay and Neal[8]. Despite their simple construction they have excellent performance. Recent improvements of low density parity check codes have allowed them to surpass the performance of turbo codes. LDPC codes are defined in terms of a sparse parity check matrix, in which most of the entries are zero and only a small fraction are nonzero values. Each code word satisfies a number of linear constraints and each symbol of the codeword participates in a small number of constraints. The constructions, description of an iterative probabilistic decoding algorithm and theory provided by Gallager goes beyond what is known today for turbo codes. Arriving before the computing power that was to
prove their effectiveness they were largely forgotten until the rediscovery by Mackay and Neal [8].

Recent advancements to LDPC codes include improvements in terms of non binary versions of the codes and codes having variable number of non zero values in the parity check matrix [10]. The non binary version of LDPC codes involved encoding the messages using symbols from a finite field with more than two elements resulting in each parity check becoming complex but decoding remaining tractable. The non binary codes have an alternative representation as binary codes but the decoding algorithm is not equivalent to the binary algorithm. LDPC codes with variable number of non zero values in the parity check matrix are known as irregular LDPC codes they have a variety of row and column weights. In irregular LDPC codes the high weight columns help the decoder to identify some errors quickly, making the remaining errors easier to correct.

The details of encoding and decoding for regular binary and non binary LDPC codes are explained in this chapter. A sparse parity check matrix $H$ is used to define a low-density parity check code. An $(N,j,k)$ low density code is a code of block length $N$ having a $(N - K) \times N$ parity check matrix $H$ with each column containing a fixed number $j$ of ones and each row containing a fixed number $k$ of ones. Figure 3.1 shows a $(20,3,4)$ code constructed by Gallager [6]. If all the rows are linearly independent the code rate is given by $R = 1 - \frac{j}{k}$, otherwise the code rate is $\frac{N - j'}{N}$ where $j'$ is the dimension of the row space of $H$. These are linear codes that use a generator matrix $G$ to map a message $m$ of length $K$ to transmitted codewords of length $N$. All codewords satisfy $nH^T = 0$, where $n$ is the assumed channel noise.
A $j$-regular LDPC code is one in which the column weight (number of ones) for each column is exactly $j$ resulting in an average row weight of $Nj/(N-K)$. The row weight can be fixed to be equal to $k = Nj/(N-K)$. The $(N,K)$ encoder accepts $K$ bit input and produces $N$-bit codeword.

### 3.3 Encoding and Decoding

The Generator matrix $G$ has to be specified for the encoder. To do this the $H$ matrix is first reduced to the systematic form $H_{sys} = [I_{N-K} / P]$ by using Gaussian elimination and column reordering. If $H$ is full rank then $H_{sys}$ will have $M = N-K$ rows, if some of the rows of $H$ are linearly independent then $H$ is not full rank and will be systematic although with fewer rows. Once $H$ is in systematic form, it is easy to confirm that a valid (systematic) generator matrix is $G_{sys} = [P^T / I_K]$ since $GH^T = 0$. The $(N, K)$ block encoder accepts the message of length $K$ and produces a codeword of length $N$. 

---

Figure 3.1 Example of a parity check matrix for $N=20$, $j=3$ and $k=4$
Figure 3.2 Message passing on the bipartite graph representing a parity-check matrix

The decoding of the codeword is done using the message passing or belief propagation or sum-product algorithm for decoding $rH^T = S \mod 2$. Elements corresponding to each row of $S_M$ are referred to as checks. Assuming the set of variables and checks as making up a belief network or Bayesian network [11] in which every variable is the parent of checks, and each check is the child of variable (figure 3.2). The network of checks and variables forms a bipartite graph defined by the $H$ matrix with variables connected only to checks and checks connected only to variables. The aim of the algorithm is to compute the marginal posterior probabilities from the observed checks. This is done by estimating the posterior probability of the value of each variable node given the received signal and the channel properties. The process can be viewed as a message passing algorithm on the bipartite graph with two sets of nodes. Nodes $S_i$ and $r_j$ are connected if the corresponding entry in the matrix $H_{ij}$ is nonzero. The directed edge
shows the causal relationship that the state of the check node is determined by the state of the variable nodes to which it is connected. At each step of the decoding algorithm each variable node \( r_j \) sends messages \( q_{mn}^x \) to each child \( S_i \) which are supposed to approximate the node’s belief that it is in state \( x \), given messages received from all its other children. Also each check \( S_i \) sends messages \( r_{mn}^x \) to each parent \( r_j \) approximating the probability of check \( i \) being satisfied if the parent is assumed to be in state \( x \), taking into account messages received from all its other parents. The messages are observed after each step and a tentative decoding is estimated. The decoding algorithm is then updated iteratively until the tentative decoding satisfies the observed syndrome vector upon which a success is declared. If the observed syndrome is not satisfied after a preset maximum number of iterations are reached failure is declared. The preset maximum number of iterations may be set to about ten times the typical number to improve the success rate while imposing little overhead on the average decoding time. In practice the algorithm usually converges and all decoding errors are detected. Sometimes the algorithm may fail to converge due to many cycles in the graph so care has to be taken to avoid many cycles but in practice the decoding performance is not affected much due to cycles.

Consider the following \( H \) matrix is an LDPC matrix with column weight \( j=2 \). This matrix represents a set of linear homogeneous modulo 2 equations called parity check equations with the set of codewords as solutions to these equations. The set of digits contained in a parity check equation is known as parity check set. Using parity check codes makes coding relatively simple to implement.
This serves as a starting point for the construction of the decoder. The matrix multiplication of the above expression defines a set of parity checks, which are

\[ p_0 = c_0 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 \\
 p_1 = c_1 \oplus c_3 \oplus c_5 \\
 p_2 = c_1 \oplus c_2 \oplus \ldots \\
 p_3 = c_0 \oplus c_2 \oplus c_4 \oplus c_6 \]

Having encoded our source vector \( m \) of length \( K \) using the generator matrix we transmit the codeword \( c = mG \). The received vector is \( r = c + n \), where \( n \) is the noise introduced by the channel. On multiplying the received vector by the \( H \) matrix we get the syndrome vector \( S = rH^T = (c + n)H^T = mGH^T + H^T n = H^T n \), with the addition and multiplication performed over a finite field. The decoding involves finding the most probable vector \( \hat{X} \) according to the channel model such that \( \hat{X}H^T = S \).
The set of variable nodes $n$ that participate in check $m$ are denoted by $N(m) = \{ n : H_{mn} = 1 \}$. We also define the set of checks in which variable node $n$ participates, $M(n) = \{ m : H_{mn} = 1 \}$. $N(m) \setminus n$ denotes the set of variable nodes excluding the variable node $n$.

There are two quantities $q_{mn}$ and $r_{mn}$ associated with each nonzero element in $H$ matrix that are alternatively updated iteratively. $q_{mn}^a$ is the probability that variable node $n$ of $X$ has the value $a$, given information obtained via checks other than the check $m$. $r_{mn}^a$ is the probability of check $m$ being satisfied if bit $n$ of $X$ is considered fixed at $a$ with the other bits having a separable distribution given by $\{ q_{m'n} : n' \in N(m) \setminus n \}$. The exact posterior probabilities of all the bits are produced after a fixed number of iterations if the bipartite graph defined by the $H$ matrix has no cycles.

Initialization:

Applying the LDPC codes to a binary symmetric channel (BSC) with crossover probability $p$.

$$p(r_j / x_j = 0) = p^{r_j} (1 - p)^{1-r_j}$$

$$p(r_j / x_j = 1) = p^{1-r_j} (1 - p)^{r_j}$$

$p(r_j / x_j = 0)$ is the prior probability of bit $r_j$ given the bit $x_j = 0$. $p(r_j / x_j = 1)$ is the prior probability of bit $r_j$ given bit $x_j = 1$.

Horizontal step:

The horizontal step involves running through the checks $m$ and compute for each $n \in N(m)$ two probabilities $r_{mn}^0$, the probability of the observed value of $S_m$ when $x_n = 0$ given
the other bits \( \{x_n : n' \neq n\} \) have a separable distribution given by the probabilities \( \{q_{mn}^0, q_{mn}^1\} \), which is obtained by running through the checks \( m \) for each \( n \in N(m) \).

\[
r_{mn}^0 = \sum_{\{x_n : n \in N(m) \setminus \{n\}\}} P(S_m \mid x_n = 0, \{x_n : n' \in N(m) \setminus \{n\}\}) \times \prod_{n \in N(m) \setminus \{n\}} q_{mn}^x.
\]

\( r_{mn}^1 \) is the probability of the observed value of \( S_m \) arising when \( x_n = 1 \), defined by

\[
r_{mn}^1 = \sum_{\{x_n : n \in N(m) \setminus \{n\}\}} P(S_m \mid x_n = 1, \{x_n : n' \in N(m) \setminus \{n\}\}) \times \prod_{n \in N(m) \setminus \{n\}} q_{mn}^x.
\]

The conditional probabilities in the summations are either zero or one, depending on whether the observed \( z_m \) matches the hypothesized values of \( x_n \) and the \( \{x_n\} \).

A convenient way to implement these probabilities is using the forward and backward passes with the products of the differences \( \delta_{mn} = q_{mn}^0 - q_{mn}^1 \) computed.

\[
\delta r_{mn} = r_{mn}^0 - r_{mn}^1
\]

\[
\delta r_{mn} = (-1)^{s_m} \prod_{n \in N(m) \setminus \{n\}} \delta q_{mn}^x
\]

Also \( r_{mn}^0 + r_{mn}^1 = 1 \), and hence \( r_{mn}^0 = (1 + \delta r_{mn}) / 2 \) and \( r_{mn}^0 = (1 - \delta r_{mn}) / 2 \)

**Vertical Step:**

In the vertical step the values \( r_{mn}^0 \) and \( r_{mn}^1 \) are used to update the probabilities \( q_{mn}^0 \) and \( q_{mn}^1 \)

\[
q_{mn}^0 = \alpha_{mn} p_{mn}^0 \prod_{m \in M(n) \setminus \{n\}} r_{m,n}^0
\]
\[
q^{1}_{mn} = \alpha_{mn} p^{1}_{n} \prod_{m \in \mathcal{M}(n) \setminus m} r^{1}_{mn}
\]

\(\alpha_{mn}\) is chosen such that \(q^{0}_{mn} + q^{1}_{mn} = 1\). These products can be efficiently computed in a downward pass and an upward pass.

Also the pseudoposterior probabilities \(q^{0}_{n}\) and \(q^{1}_{n}\) at this iteration are computed using

\[
q^{0}_{n} = \alpha_{n} p^{0}_{n} \prod_{m \in \mathcal{M}(n)} r^{0}_{mn}
\]

\[
q^{1}_{n} = \alpha_{n} p^{1}_{n} \prod_{m \in \mathcal{M}(n)} r^{1}_{mn}
\]

These quantities are used to create a tentative decoding \(\hat{X}\), this is used to calculate the syndrome using the \(H\) matrix. If the syndrome is satisfied the decoding is halted and success is declared otherwise the decoding is iterated for a present maximum number of iterations. If the syndrome is not satisfied even after the maximum number of iterations are reached a failure is declared.

The difference between this decoding algorithm and the decoding algorithm used in turbo codes is that the decoding algorithm in turbo codes is run for the preset maximum number of iterations irrespective of whether a consistent state is found or not. This results in wastage of computer time. In the decoding algorithm used in LDPC codes undetected error occurs only if the \(\hat{X}\) estimated satisfies \(\hat{X}H^T = S \mod 2\) and is not the true \(X\). Detected errors occur if the maximum number of iterations is completed without valid decoding.
3.4 Non binary LDPC Codes

The binary low density parity check codes represented by sparse binary parity check matrices and corresponding bipartite graphs can be generalized by using the same bipartite graphs. For the generalized LDPC codes the variable nodes can take values from some finite alphabet and the check nodes impose constraints more complex than binary parity checks [12].

A vector space over the finite field $GF(q)$ where $q = 2^b$ can be used to generalize the binary LDPC codes in a natural way. We choose powers of two so that we can continue to transmit using binary channels, transmitting $q$-ary symbols for every $b$ uses of the channel. Elements of $GF(q)$ are called as symbols. A symbol from the field $GF(2^b)$ may be represented as a binary string of $b$ bits. Weight of a vector is the number of nonzero symbols in it. A very sparse random parity check matrix $H$ is used to define the non binary LDPC code with a transmitted block length $N$ and a source block length $K$. The nonzero entries in each row of $H$ are chosen to maximize the entropy of the corresponding symbol of the syndrome vector $S = xH^T$, $x$ is a sample from the assumed channel noise model.

Decoding

The decoding algorithm is the generalization of the approximate belief propagation algorithm. The elements of $X$ are variables and the elements of $S$ are checks. $N(m) = \{n : H_{mn} \neq 0\}$ denotes the set of variables participating in the check $m$. $M(n) = \{m : H_{mn} \neq 0\}$ denotes the set of checks that depend on variable symbol $n$. Quantities $q_{mn}^a$ and $r_{mn}^a$ are calculated for each nonzero entry in the $H$ matrix. $q_{mn}^a$ is the probability that the symbol $n$ of $x$ is $a$ given the information obtained from checks other than the check $m$. $r_{mn}^a$
is the probability of check \( m \) being satisfied given that symbol \( n \) of \( x \) is fixed at \( a \) with the other variable symbols having a separable distribution given by \( \{ q_{mn}^{a} : n \in N(m) \setminus n, a \in GF(q) \} \).

Initialization:

Applying the codes to binary symmetric channel (BSC) with cross over probability \( p \), we define \( g_{n}^{1} = p^{1-r_{n}}(1-p)^{r_{n}} \) and \( g_{n}^{0} = 1 - g_{n}^{1}. \) \( g_{n} \) is independent of \( n \).

The channel likelihoods are set to 
\[
    f_{n}^{a} = \prod_{i=1}^{b} g_{n_{i}}^{a_{i}} \text{ for each noise symbol } x_{n} \text{ being equal to } a, \text{ for each } a \in GF(q), \text{ } a_{i} \text{ is the binary representation of } a. \text{ Where each noise symbol } x_{n} \text{ consists of } b \text{ bits } x_{n_{0}}...x_{n_{b}} \text{ and } g_{n_{i}}^{a_{i}} \text{ is the likelihood of the } i^{th} \text{ constituent bit } (x_{i}) \text{ is equal to } a_{i}, \text{ where } (a_{1}...a_{p}) \text{ is the binary representation of the symbol } a.
\]

Horizontal step:

Horizontal step is the computation of the quantity \( r_{mn}^{a} \) which is the probability of check \( m \) being satisfied if symbol \( n \) on \( x \) is considered fixed at \( a \) and other noise symbols have a separable distribution given by \( \{ q_{mn}^{a} : n \in N(m) \setminus n, a \in GF(q) \} \).

Value of \( r_{mn}^{a} \) is computed using

\[
    r_{mn}^{a} = \sum_{x_{n} = a} P[S_{m} | x_{n} = a] \times \prod_{j \in N(m) \setminus n} q_{mj}^{x_{j}}
\]

Prob\( [S_{m} / x_{n} = a] \) is 0 or 1 depending on whether or not \( x_{n} \) satisfies the check \( m. \) \( r_{mn}^{a} \) can be calculated efficiently by defining partial sums \( \sigma_{mk} := \sum_{j : j \leq k} H_{mj} x_{j} \) and \( \rho_{mk} := \sum_{j : j \geq k} H_{mj} x_{j} \) and calculating \( \Pr[\sigma_{mk} = a] \) for each \( a \in GF(q) \) and each \( k \in N(m) \) according to the probabilities given by \( q. \) If \( i,j \) are successive indexes in \( N(m) \) and \( j > i \) then
\[
\Pr[\sigma_{mj} = a] = \sum_{\{s,t: H_{m,t}^+ s = a\}} \Pr[\sigma_{mj} = s] \Pr[\rho_{mj}]
\]

On the same lines we can also calculate the distribution of each \( \rho_{mk} \). Using \( \sigma_{mk} \) and \( \rho_{mk} \) the quantity \( r^a_{mn} \) can be updated using the below equation

\[
r^a_{mn} = \frac{\Pr[(\sigma_{m(n-1)} + \rho_{m(n+1)}) = S_m - H_{mn} a]}{\Pr[\sigma_{m(n-1)} = s] \Pr[\rho_{m(n+1)} = t]}
\]

Vertical Step:

Vertical step involves updating the \( q^a_{mn} \) which is the probability that the symbol \( n \) on \( x \) is \( a \), given information obtained from checks other than check \( m \).

Value of \( q^a_{mn} \) is updated for each \( m \) and \( n \) using

\[
q^a_{mn} = \alpha_{mn} f^a_n \prod_{j \in M(n)\setminus m} r^a_{jn} \alpha_{mn} \text{ is chosen such that } \sum_{a=1}^{q} q^a_{mn} = 1
\]

Tentative decoding \( \hat{x} \) is made using \( \hat{x}_n = \arg \max_a f^a_n \prod_{j \in M(n)} r^a_{jn} \)

If a valid decoding of the syndrome \( \hat{x}H = S \) is identified then the algorithm halts, else the algorithm is repeated. Valid decoding of the syndrome is an all zero syndrome. Failure is declared after some maximum number of iterations a valid decoding is not obtained.

The advantages of the generalized low density parity check codes are that in the generalized \( GF(q) \) field the mean column weight \( t \) of the equivalent binary parity check matrix is increased while retaining the same bipartite graph to perform decoding. Also on comparing the graphs (figure 3.3) of two equivalent matrix fragments the \( q \)-ary code contains no cycles where as binary code has a cycles.
Another difference between binary and $q$-ary codes is that the state space of each node is increased in the decoding graph by decoding over $GF(q)$. This allows us to track correlations in the true posterior distribution that are not detectable by the binary algorithm. Also increasing the field order $q$ for LDPC codes is comparable to increasing the memory of convolutional codes. The only drawback of moving to $GF(q)$ is the increase in decoding complexity.
4. DISTRIBUTED SOURCE CODING USING BINARY LOW DENSITY PARITY CHECK CODES

Low density parity check codes can be used for applications involving compression of two correlated sources using the syndrome concept. The compressed sequence of the source output bits is the syndrome, which is determined using the parity check matrix \( H \).

It has been shown [2] that LDPC codes can be employed when viewing the problem using an equivalent channel and applying the syndrome approach for the case where one of the two correlated sources is available losslessly at the joint decoder. This can be viewed as application of LDPC codes to a compression problem with side information. It is based on modifying the conventional message passing LDPC decoder to take into account the syndrome information. Also all LDPC code design techniques can be applied to distributed source coding producing simulation results better than any turbo coding scheme.

![Figure 4.1 Modeling Correlation using BSC](image)

The system considered consists of two equiprobable binary random sources \( X = [X_1, X_2 \ldots X_n] \) and \( Y = [Y_1, Y_2 \ldots Y_n] \). \( X_i \) and \( Y_i \) are correlated with \( \Pr[X_i \neq Y_i] = p < 0.5 \). The correlation between \( X_i \) and \( Y_i \) is modeled using a binary symmetric channel with crossover probability \( p \). \( X_i \) being the input to the channel and \( Y_i \) being the output (figure 4.1), with the
compressed version of $X$ looking like a codeword to the channel. The source $Y$ is available losslessly at the joint decoder and source $X$ is compressed. The rate used for $Y$ is its entropy $NR_2 = NH(Y_i) = N\text{bits}$, therefore the theoretical limit for lossless compression of $X$ is \cite{1} $nR_1 \geq nH(X_i/Y_i) = nH(p) = n(-p \log_2 p - (1-p) \log_2(1-p))$. The compression results in mapping a sequence of $N$ input bits into $N-K$ syndrome bits resulting in a compression ratio of $N : (N-K)$, known as the “Wyner’s scheme” \cite{13}. The all zero syndrome of the linear block code is considered to be the original code for distributed coding.

4.1 Encoding and Decoding with Binary LDPC Codes

Encoding

Encoding is done by multiplying the source $X$ [length $N$] by the parity check matrix $H$, resulting in the compressed $X$ which is the syndrome $S$ [length $(N-K)$].

$$S = X \times H^T$$

In the bipartite graph encoding can be viewed as the binary addition of all the variable nodes connected to the same check node.

Considering an example of $H$ matrix as shown below

$$H = \begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1
\end{bmatrix}$$
Here the vector $S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix}$ is the compressed version of the source sequence $X$. As the vector $S$ is binary it consists of ones and zeros. When the value of $S_i$ is zero the parity check equation obtained by multiplying the source $X$ [length $N$] by the parity check matrix $H$ is same as the parity check equation of low density parity check codes, which is obtained by multiplying the received vector with the $H$ matrix for channel coding. When the value of $S_i$ is one the parity check equation obtained by multiplying the source $X$ [length $N$] has to satisfy value of $S_i$. This is taken care of in the horizontal step of the decoding algorithm.

**Decoding**

Decoding involves the estimation of the $N$-length sequence $X$ from the $(N-K)$ length syndrome $S$ and the $N$-length sequence $Y$. The decoding algorithm is similar to LDPC decoding used for channel coding except for the inclusion of the syndrome bits in the horizontal step of the algorithm.
The set of noise bits $n$ that participate in check $m$ are denoted by $N(m) = \{ n : H_{mn} = 1 \}$. We also define the set of checks in which noise bit $n$ participates, $M(n) = \{ m : H_{mn} = 1 \}$. $N(m) \setminus n$ denotes the set of noise bits excluding the noise bit $n$. There are two quantities $q_{mn}$ and $r_{mn}$ associated with each nonzero element in $H$ matrix that are alternatively updated iteratively. $q_{mn}^a$ is the probability that noise bit $n$ of $X$ has the value $a$, given information obtained via checks other than the check $m$. $r_{mn}^a$ is the probability of check $m$ being satisfied if bit $n$ of $X$ is considered fixed at $a$ with the other bits having a separable distribution given by $\{ q_{mn} : n' \in N(m) \setminus n \}$.

Initialization:

Considering a binary symmetric channel (BSC) with crossover probability $p$. We model the correlation between the two sources using the binary symmetric channel with $X_i$ as the input to the channel and $Y_i$ as the output. The compressed version of $X$ is the syndrome $S$ which is made to look like a codeword of the channel.

\[
p(y_j | x_j = 0) = p^{y_j} (1 - p)^{1-y_j},
\]

\[
p(y_j | x_j = 1) = p^{1-y_j} (1 - p)^{y_j},
\]

Horizontal step:

The horizontal step involves the computation of two probabilities $r_{mn}^0$ and $r_{mn}^1$, the probability of the observed values of $S_m$ when $x_n = 0$ and $x_n = 1$ respectively, given the other bits $\{ x_{n'} : n' \neq n \}$ have a separable distribution given by the probabilities $\{ q_{mn}^0, q_{mn}^1 \}$, which is obtained by running through the checks $m$ for each $n \in N(m)$. The syndrome information is included in this step to modify the message passing LDPC decoder for distributed source coding.
\( r_{mn}^0 \) is the probability of the observed value of \( S_m \) arising when \( x_n = 0 \), defined by

\[
r_{mn}^0 = \sum_{\{x_n : n \in N(m) \setminus n\}} P(S_m / x_n = 0, \{x_n : n \in N(m) \setminus n\}) \times (1 - 2S_m) \prod_{n \in N(m) \setminus n} q_{mn}^x.
\]

\( r_{mn}^1 \) is the probability of the observed value of \( S_m \) arising when \( x_n = 1 \), defined by

\[
r_{mn}^1 = \sum_{\{x_n : n \in N(m) \setminus n\}} P(S_m / x_n = 1, \{x_n : n \in N(m) \setminus n\}) \times (1 - 2S_m) \prod_{n \in N(m) \setminus n} q_{mn}^x.
\]

The conditional probabilities in the summations are either zero or one, depending on whether the observed \( z_m \) matches the hypothesized values of \( x_n \) and the \( \{x_n\} \).

A convenient way to implement these probabilities is using the forward and backward passes with the products of the differences \( \delta q_{mn}^0 = q_{mn}^0 - q_{mn}^1 \) computed.

\[
\delta r_{mn} = r_{mn}^0 - r_{mn}^1,
\]

\[
\delta r_{mn} = (-1)^{S_m} \prod_{n \in N(m) \setminus n} \delta q_{mn}.
\]

Also \( r_{mn}^0 + r_{mn}^1 = 1 \), and hence \( r_{mn}^0 = (1 + (1 - 2S_m)\delta r_{mn}) / 2 \) and \( r_{mn}^0 = (1 - (1 - 2S_m)\delta r_{mn}) / 2 \).

The syndrome which is the compressed information is included in the horizontal step in the calculation of \( r_{mn}^0 = (1 + (1 - 2S_m)\delta r_{mn}) / 2 \) and \( r_{mn}^0 = (1 - (1 - 2S_m)\delta r_{mn}) / 2 \).

Vertical Step:

In the vertical step the values \( r_{mn}^0 \) and \( r_{mn}^1 \) are used to update the probabilities \( q_{mn}^0 \) and \( q_{mn}^1 \)

\[
q_{mn}^0 = \alpha_{mn} P_{n}^0 \prod_{m \in M(n) \setminus m} r_{mn}^0.
\]
\[ q_{mn}^1 = \alpha_{mn} p_n^1 \prod_{m \in M(n)} r_{mn}^1 \]

\( \alpha_{mn} \) is chosen such that \( q_{mn}^0 + q_{mn}^1 = 1 \). These products can be efficiently computed in a downward pass and an upward pass.

Also the pseudoposterior probabilities \( q_n^0 \) and \( q_n^1 \) at this iteration are computed using

\[ q_n^0 = \alpha_n p_n^0 \prod_{m \in M(n)} r_{mn}^0 \]

\[ q_n^1 = \alpha_n p_n^1 \prod_{m \in M(n)} r_{mn}^1 \]

These quantities are used in the estimation of \( \hat{X} \), which is the \( N \)-length estimate of \( X \).

The horizontal and vertical steps are repeated for a given number of iterations after which the estimate \( \hat{X} \) of the \( N \)-length message vector \( X \) is determined. The estimated \( \hat{X} \) is then compared with \( X \) and the bit error rate is calculated.

The main difference in the decoding of LDPC for distributed source coding and channel coding is that in case of channel coding the message passing of the probabilities between the horizontal and vertical step is stopped on the occurrence of a zero syndrome vector. The syndrome in case of channel coding is the decoded message multiplied with the parity check matrix. In distributed source coding iterations are fixed to a given value. The decoding is carried out for the fixed number of iterations after which the message vector \( \hat{X} \) is estimated. If the estimate \( \hat{X} \) is not the same as the original message \( X \) then the number of iterations for the decoding algorithm is increased. In the above implementation the decoding algorithm has been implemented in the probability domain.
5. DISTRIBUTED SOURCE CODING USING NON BINARY LDPC

Generalized non binary low density parity check codes can be used for applications involving compression of two correlated non binary sources using the syndrome concept. The compressed sequence of the source output bits is the syndrome, which is determined using the parity check matrix $H$. The non binary LDPC algorithm has been modified to take into account the syndrome information during decoding.

The system considered consists of two non binary information sources $X = [X_1, X_2, \ldots, X_n]$ and $Y = [Y_1, Y_2, \ldots, Y_n]$. $X_i$ and $Y_i$ are correlated with $\Pr[X_i \neq Y_i] = p < 0.5$. The correlation between $X_i$ and $Y_i$ is modeled using a binary symmetric channel with crossover probability $p$. $X_i$ being the input to the channel and $Y_i$ being the output, with the compressed version of $X$ looking like a codeword to the channel. The source $Y$ is available losslessly at the joint decoder and source $X$ is compressed. The rate used for $Y$ is its entropy $NR_2 = NH(Y_i) = N_{\text{symbols}}$, therefore the theoretical limit for lossless compression of $X$ is [1] $nR_1 \geq nH(X_i/Y_i) = nH(p) = n(-p \log_2 p -(1-p) \log_2 (1-p))$. The compression results in mapping a sequence of $N$ input symbols into $N-K$ syndrome symbols resulting in a compression ratio of $N : (N - K)$, known as the “Wyner’s scheme” [13].

5.1 Encoding and Decoding with Non Binary LDPC Codes

Encoding

Encoding is done by multiplying the source $X$ [length $N$] by the parity check matrix $H$, resulting in the compressed $X$ which is the syndrome $S$ [length $(N-K)$]. By

$$S = X \times H^T$$
In the bipartite graph encoding can be viewed as the addition of all the variable nodes connected to the same check node in the generalized Galois field.

Considering an example of $H$ matrix of the GF (4) field as shown below

$$H = \begin{bmatrix} 
3 & 0 & 0 & 2 & 1 & 1 & 3 \\
0 & 2 & 0 & 2 & 0 & 3 & 0 \\
0 & 1 & 3 & 0 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 3 & 0 & 3 
\end{bmatrix}$$

$$[X_0, X_1, X_2, X_3, X_4, X_5, X_6] \begin{bmatrix}
3 & 0 & 0 & 2 \\
0 & 2 & 1 & 0 \\
0 & 0 & 3 & 1 \\
2 & 2 & 0 & 0 \\
1 & 0 & 0 & 3 \\
1 & 3 & 0 & 0 \\
3 & 0 & 0 & 3 
\end{bmatrix} = \begin{bmatrix} 
S_0 \\
S_1 \\
S_2 \\
S_3 
\end{bmatrix}$$

$$3X_0 + 2X_3 + X_4 + X_5 + X_6 = S_0$$
$$2X_1 + 2X_3 + 3X_5 = S_1$$
$$X_1 + 3X_2 = S_2$$
$$2X_0 + X_2 + 3X_4 + 3X_6 = S_3$$

For channel coding for the decoding algorithm to halt the vector $S= \begin{bmatrix} S_0 \\
S_1 \\
S_2 \\
S_3 
\end{bmatrix}$, obtained by multiplying the received vector with the $H$ matrix should be an all zero vector which indicates the correct decoding of the message from the encoded codeword. Where as for source coding the vector $S= \begin{bmatrix} S_0 \\
S_1 \\
S_2 \\
S_3 
\end{bmatrix}$ represents the compressed information using which the decoder with the help of the side information should decode the original source vector $X$. So the decoding algorithm must
use this syndrome information. This has to be taken into consideration in the implementation of the decoding algorithm. The horizontal step of the decoding algorithm is modified to include the syndrome information.

Decoding

The decoding algorithm is same as the decoding algorithm of LDPC for channel coding it is the generalization of the approximate belief propagation algorithm. The elements of $x$ are variable symbols and the elements of $S$ are checks. $N(m) = \{ n : H_{mn} \neq 0 \}$ denotes the set of variable symbols participating in the check $m$. $M(n) = \{ m : H_{mn} \neq 0 \}$ denotes the set of checks that depend on variable symbol $n$. Quantities $q_{mn}^a$ and $r_{mn}^a$ are calculated for each nonzero entry in the $H$ matrix. $q_{mn}^a$ is the probability that the symbol $n$ of $x$ is a given the information obtained from checks other than the check $m$. $r_{mn}^a$ is the probability of check $m$ being satisfied given that symbol $n$ of $x$ is fixed at $a$ with the other variable symbols having a separable distribution given by $\{ q_{mn}^a : n \in N(m) \setminus a, a \in GF(q) \}$. Updating of $r_{mn}^a$ uses the syndrome, which is the compressed information thereby, making the decoding algorithm of non binary LDPC to implement distributed source coding.

Initialization:

The initialization is similar to initialization of LDPC for channel coding the only difference being that the side information is initialized based on the channel probabilities. Applying the codes to binary symmetric channel (BSC) with cross over probability $p$, we define $g_n^1 = p^{1-r_f} (1-p)^r_f$ and $g_n^0 = 1 - g_n^1$, $g_n^1$ is independent of $n$. 
The channel probabilities are set to 
\[ f_n^a = \prod_{i=1}^{b} g_{n_i}^{a_i} \] for each variable symbol \( y_n \) being equal to \( a \), for each \( a \in GF(q) \), \( a_i \) is the binary representation of \( a \). Where each variable symbol \( y_n \) consists of \( b \) bits \( y_{n_1} \ldots y_{n_b} \) and \( g_{n_i}^{a_i} \) is the likelihood of the \( i^{th} \) constituent bit \( (y_i) \) is equal to \( a_i \), where \( (a_1 \ldots a_p) \) is the binary representation of the symbol \( a \).

The initialization can also be done on a \( q \)-ary symmetric channel where in the side information is initialized based on \( q \)-ary symmetric channel probabilities. Applying the codes to a \( q \)-ary symmetric channel (QSC) with cross over probability \( p \), we define 
\[ f_n^a = 1 - p \] and 
\[ f_n^b = \frac{p}{q-1} \] where \( a \neq b \).

Horizontal step:

Horizontal step is the computation of the quantity \( r_{mn}^a \) which is the probability of check \( m \) being satisfied if symbol \( n \) of \( y \) is considered fixed at \( a \) and other noise symbols have a separable distribution given by \( \{ q_{mn}^a : n \in N(m) \setminus n, a \in GF(q) \} \).

Value of \( r_{mn}^a \) is computed using
\[ r_{mn}^a = \sum_{y_n=a} P[S_m \mid y_j] \times \prod_{j \in N(m) \setminus n} q_{mj}^{y_j} \]
Prob\( [S_m \mid y] \) is 0 or 1 depending on whether or not \( y \) satisfies the check \( m \), which is the compressed information in distributed source coding. \( r_{mn}^a \) can be calculated efficiently by defining partial sums \( \sigma_{mk} := \sum_{j : j \leq k} H_{mj} y_j \) and \( \rho_{mk} := \sum_{j : j \geq k} H_{mj} y_j \) and calculating \( Pr[\sigma_{mk} = a] \) for each \( a \in GF(q) \) and each \( k \in N(m) \) according to the probabilities given by \( q \). If \( i,j \) are successive indexes in \( N(m) \) and \( j > i \) then
\[
\Pr[\sigma_{mj} = a] = \sum_{s, t: H_{mt} + s = a} \Pr[\sigma_{mj} = s] q_{mj}^t
\]

On the same lines we can also calculate the distribution of each \( \rho_{mk} \). Using \( \sigma_{mk} \) and \( \rho_{mk} \) the quantity \( r_{mn}^a \) can be updated using the below equation

\[
r_{mn}^a = \Pr[(\sigma_{m(n-1)} + \rho_{m(n+1)}) = S_m - H_{mn}a] \\
= \sum_{s, t: s + t = S_m - H_{mn}a} \Pr[\sigma_{m(n-1)} = s] \Pr[\rho_{m(n+1)} = t]
\]

In non binary LDPC decoding for channel coding \( s, t : s + t = S_m - H_{mn}a \) is solved such that the above equation is satisfied when \( S_m = 0 \), however in distributed source coding \( S_m \) represents the compressed information. Therefore \( s, t : s + t = S_m - H_{mn}a \) is solved for the value of the \( m^{th} \) syndrome. Considering the equation \( s, t : s + t = S_m - H_{mn}a \) for channel coding \( s, t : s + t + H_{mn}a = 0 \), as \( S_m = 0 \) but for distribute source coding \( s, t : s + t + H_{mn}a = S_m \) with \( S_m \neq 0 \) or \( s, t : s + t + H_{mn}a - S_m = 0 \).

**Vertical Step:**

Vertical step involves updating the \( q_{mn}^a \) which is the probability that the symbol \( n \) on \( x \) is \( a \), given information obtained from checks other than check \( m \).

Value of \( q_{mn}^a \) is updated for each \( m \) and \( n \) using

\[
q_{mn}^a = \alpha_{mn} f_n^a \prod_{j \in M(n) \setminus m} r_{jn}^a \quad \alpha_{mn} \text{ is chosen such that } \sum_{a=1}^q q_{mn}^a = 1
\]

Tentative decoding \( \hat{X} \) is made using \( \hat{X}_n = \arg \max_a f_n^a \prod_{j \in M(n)} r_{jn}^a \). The decoding is done for a fixed number of preset iterations after which \( \hat{X} \) the estimate is got. If the estimate \( \hat{X} \) is not the same as the original source \( X \), the number of iterations is increased and the decoding is done again.
6. SIMULATION AND RESULTS

6.1 Distribute Source Coding using Binary LDPC

Binary LDPC codes in the probability domain are used to implement distributed source coding. The implementation was done using MATLAB (see appendix). Simulations were done using two binary correlated sources consisting of 200 bits each and a compression rate of 1/2. Simulations were done for different correlations of 0.80, 0.85, 0.90, 0.95 between the two sources and the corresponding bit error rates were observed. A plot of the bit error rate versus correlation is as shown in figure 6.1.

![Bit Error Rate v/s Correlation Coeff](image)

Figure 6.1 Plot for Distributed source coding using binary LDPC

6.2 Distributed Source Coding using Non Binary LDPC

Non binary LDPC codes for distributed source coding were simulated for $GF(4)$ and $GF(8)$ non binary fields. Two correlated sources consisting of 200 symbols each were used. One
of the sources was compressed to half its original size. The correlation between the two sources was modeled using a virtual binary symmetric channel. Simulations were carried out for same correlation coefficients as given above between the two sources. A plot of symbol error rate versus correlation is as shown in figure 6.2 and 6.3 for $GF(4)$ and $GF(8)$ respectively. In addition for $GF(4)$ the correlation between the two non binary sources was modeled using a virtual $q$-ary symmetric channel. Simulations were carried out for different correlations between the two sources. Figure 6.4 shows the plot of symbol error rate/bit error rate versus correlation for the two sources consisting of 200 symbols for non binary LDPC and equivalent 400 binary bits for binary LDPC using a compression rate of 1/2.

![Symbol Error Rate vs Correlation Coeff](image)

Figure 6.2  Plot for Distributed source coding using non binary LDPC over $GF(4)$ for compression rate 1/2
Figure 6.3  Plot for Distributed source coding using non binary LDPC over $GF(8)$ for compression rate 1/2

Figure 6.4  Plot for Distributed source coding using binary and non binary LDPC over $GF(4)$ for compression rate 1/2
Simulations were also carried out for a compression rate of 3/4 for correlation coefficients of 0.70, 0.75, 0.80, 0.85, 0.90, 0.95 between the two sources. A plot of symbol error rate versus correlation is as shown in figure 6.5 and 6.6 for $GF(4)$ and $GF(8)$ respectively. Figure 6.7 shows the plot of symbol error rate/bit error rate versus correlation for the two sources consisting of 400 symbols for non binary LDPC and equivalent 800 binary bits for binary LDPC using a compression rate of 3/4.

Figure 6.5  Plot for Distributed source coding using non binary LDPC over $GF(4)$ for compression rate 3/4
Figure 6.6 Plot for Distributed source coding using non binary LDPC over $GF(8)$ for compression rate $3/4$

Figure 6.7 Plot for Distributed source coding using binary and non binary LDPC over $GF(4)$ for compression rate $3/4$
6.3 Observations and Results

The plots show that for a correlation coefficient of 0.90 and above the bit error rate and symbol error rate is zero for binary sources and non binary sources for compression rate of 1/2 when the correlation is modeled using a virtual binary symmetric channel as shown in figures 6.1, 6.2 and 6.3. For non binary sources with the correlation between the sources modeled using a \(q\)-ary symmetric channel it can be observed from figure 6.4 that for correlation coefficients of 0.80 and above the symbol error rate is zero for compression rate of 1/2. Also using a compression rate of 3/4 zero bit error rate can be obtained for correlation coefficients of 0.75 and above for binary sources and zero symbol error rate can be obtained for correlation coefficients of 0.65 and above for non binary sources as seen from figures 6.5, 6.6 and 6.7. It can be inferred that distributed source coding using LDPC can be used to recover the information from the compressed source with no symbol error rate for sources that have 0.9 or greater correlation by modeling the correlation using a virtual binary symmetric channel for compression rate of 1/2. It is possible to recover information from compressed source for correlations less than 0.9 but with some loss. When the correlation between the sources is modeled using a \(q\)-ary symmetric channel correlations of 0.80 and above also can be used to completely recover the original source. For compression rates less than 1/2 complete recovery of the original source can be obtained for lesser correlation coefficients between the two sources. Thus it is possible to use distributed source coding for transmission of correlated sources using LDPC coding.
7. CONCLUSIONS

In this thesis we have shown that non binary low-density parity-check (LDPC) codes can be used to compress correlated non binary sources modeling the correlation between the two sources using a virtual binary symmetric channel and $q$-ary symmetric channel. The scheme is implemented viewing the correlation as a channel and applying the syndrome concept.

We can observe from the results that binary LDPC in the probability domain can be used to implement distributed source coding. Also for compression of non binary sources non binary LDPC can be used, which overcomes the reduction in correlation if converted to binary bits. This scheme can be used for compression of video frames. Also from the plots it can be observed that the compressed source can be recovered without any loss for correlations greater than .9 rendering this scheme ideal for compression of video frames as video frames have very high correlation.

In the future, the proposed scheme can be extended for the compression of video frames. It can be used for practical implementation of distributed source coding in video communication and sensor networks. It can also be extended to more than two sources and for implementation in the time domain.
BIBLIOGRAPHY


Main Routine

```matlab
clear all;
clc;
q = 4;                       % Field parameter
nbits = log2(q);            % bits per symbol
n=400;
k=300;
init=ones(q,n);

h = ldpc_generate(k,n,3,q,123);  % Generate H
[H,G] = ldpc_h2g(h,q);               % find systematic G and modify H
m=size(G,2);
x= floor(rand(1,m)*q);       % random symbols
y=x;
for i=1:103
    y(i)=~x(i);
end
rho=corrcoef(x,y)
P=1-rho;
P=P(1,2);
Syndrome = ldpc_encode_dis(x,H,q);     % encoding

%%%initialization%%%%%%%%%%%%%%%%%for i=1:n
if(y(i)==0)
    init(1,i)=1-P;
    init(2,i)=P/(q-1);
    init(3,i)=P/(q-1);
    init(4,i)=P/(q-1);
elseif(y(i)==1)
    init(1,i)=P/(q-1);
    init(2,i)=1-P;
    init(3,i)=P/(q-1);
    init(4,i)=P/(q-1);
elseif(y(i)==2)
    init(1,i)=P/(q-1);
    init(2,i)=P/(q-1);
    init(3,i)=1-P;
    init(4,i)=P/(q-1);
else
    init(1,i)=P/(q-1);
    init(2,i)=P/(q-1);
    init(3,i)=P/(q-1);
    init(4,i)=1-P;
end
end
[x_hat, success, k] = ldpc_decode_dis(init,H,q,Syndrome);

[x_hat, success, k] = ldpc_decode_dis(init,H,q,Syndrome);
[differ = find(x_hat ~= x);
Bit_error = size(differ);
No_errors = Bit_error(1,2);
BER = No_errors/n
```
Encoding Function

function [out] = ldpc_encode(in,G,q)
[k,n] = size(G);
G=G';
if q==2 % binary case
    out = mod((in)*G,2);
else
    M=log2(q); % GFq exponent
    [tuple power] = gftuple([-1:2^M-2]', M, 2); %%%%equ GF represe and matlab exponen rep
    alpha = tuple * 2.^[0 : M - 1]; %%%%back to gf(q) format
    beta(alpha + 1) = 0 : 2^M - 1; %%%%help store in matlab
    ll = ones(1,n)*(-Inf); % will store results here, initialize with all zeros (exp form)
    for i=1:k % multiply each row of G by the input symbol in GFq
        ii = power(beta(in(i)+1)+1); % get expon. representation of in(i)
        jj = power(beta(G(i,:)+1)+1); % same for the row of G
        kk = gfadd(power(ii,jj,tuple)); % this is exponential representation of the product
        ll = gfadd(ll,kk,tuple);
    end
    out=zeros(size(ll));
    nzindx = find(isfinite(ll));
    out(nzindx) = alpha(ll(nzindx)+2);
    out = out(:);
end

Decoding Function

function [x_hat, success, k] = ldpc_decode(f,H,qq,z)
% GFq, nonbinary
    [m,n] = size(H); if m>n, H=H'; [m,n] = size(H); end
    if ~issparse(H) % make H sparse if it is not sparse yet
        [ii, jj, sH] = find(H);
        H = sparse(ii, jj, sH, m, n);
    end
% initialization
    [ii, jj, sH] = find(H); % subscript index to nonzero elements of H
    W = sparse(ii, jj, ones(size(ii)), m, n); % indicator function
    nvars = full(sum(W, 2)); % number of variables participating each check function
    minvars = min(nvars); % min number of variables in a function
    maxvars = max(nvars); % max number of variables in a function
    nfuns = full(sum(W, 1)); % number of functions per variable
    minfuns = min(nfuns); % min number of functions per variable
    maxfuns = max(nfuns); % max number of functions per variable
    % the following will be used in solving linear equations over GFq
    M=log2(qq); % GFq exponent
    [tuple power] = gftuple([-1:2^M-2]', M, 2); %%%%equ GF represe and matlab exponen rep
    alpha = tuple * 2.^[0 : M - 1];
    beta(alpha + 1) = 0 : 2^M - 1;
    % create cell arrays which contain sparse matrices with fixed # of variables in rows
    for nvars = minvars:maxvars
        tmp = zeros(size(H));
        rows = find(nvars == nvars); % rows of H having 'nnvars' variables
tmp(rows,:) = H(rows,:);  
[ijj,iii,ssH] = find(tmp');
ii{nnvars} = reshape(ijj,nnvars,length(ijj)/nnvars)';
jjr{nnvars} = reshape(jjj,nnvars,length(jjj)/nnvars)';
Hr{nnvars} = reshape(ssH,nnvars,length(ssH)/nnvars);  % separate parity matrices
q{nnvars} = reshape(f(:,jjr{nnvars}'),[size(jjr{nnvars}1,qq)];  %initialize to channel likelihoods
% Prestore valid configurations in array X
if(~isempty(Hr{nnvars}))  % make sure the are functions for this case
Hleft = Hr{nnvars}(1,:);         % will solve for these variables
Hright = Hr{nnvars}(:,2:nnvars);  % while setting these arbitrary
for i=0:(qq^(nnvars-1)-1)  % there are qq^(nnvars-1) different combinations
xr = (fliplr(de2bi(i,nnvars-1,qq)));  % current nonzero combination
% find the remaining variable to satisfy the parity checks
right_part = ones(size(Hleft))*(-Inf);  % exponent over GFq
for j=1:(nnvars-1)  % multiply each column of Hright by the symbol from x and accumulate
rr1 = power(beta(xr(j)+1)+1);      % get expon. representation of xr(i)
rr2 = power(beta(Hright(:,j)+1)+1);  % same for the column of Hright
rr3 = gfmul1(rr1,rr2,tuple);  % this is exponential representation of the product
end
left_part= mod(gfadd(right_part,power(beta(z+1)+1),tuple) - power(beta(Hleft+1)+1),qq-1);
xl=zeros(size(left_part));
[nzindx] = find(isfinite(left_part));
xl(nzindx) = alpha(left_part(nzindx)+2);
x = [xl repmat(xr,[length(xl),1])];  % this is a valid configuration
X{nnvars}(i+1,:,:) = x;
end

% create cell arrays which contain sparse matrices with fixed # of functions in columns
for nnfuns = minfuns:maxfuns
    tmp = zeros(size(H));
    cols = find(nnfuns == nnfuns);  % rows of H having 'nnvars' variables
    tmp(:,cols) = H(:,cols);
    [iii,iij,ssH] = find(tmp);
    iiic{nnfuns} = reshape(iii,nnfuns,length(iii)/nnfuns);
    jjc{nnfuns} = reshape(jjj,nnfuns,length(jjj)/nnfuns);
    Hc{nnfuns} = reshape(ssH,nnfuns,length(ssH)/nnfuns);  % separate parity matrices
    if{nnfuns} = reshape(f(:,jjc{nnfuns}'),[size(jjc{nnfuns}1,qq)];  % this will not change
end

% iterations
k=0;
success = 0;
max_iter = 100;
while ((success == 0) & (k < max_iter)),
   k = k+1;
   buffer = zeros([size(H),qq]);
   % Horizontal step - forming messages to variables from the parity check functions
   % each Hr is processed separately
   for nnvars = minvars:maxvars
      if(~isempty(Hr{nnvars}))  % make sure the are functions for this case
         result = zeros([size(Hr{nnvars}1,qq)];  % will store the intermediate result
         for i=0:(qq^(nnvars-1)-1)  % there are qq^(nnvars-1) different combinations
            x = squeeze(X{nnvars}(i+1,:,:));  % lookup a valid configuration
            % calculate products
            a = cumsum(ones(size(x),1));
            % iterations
            ...
b = cumsum(ones(size(x),2));
idx = sub2ind(size(q{nnvars}),a,b,x+1);  %index of current configuration in 3D
pp = repmat(prod(q{nnvars}(idx),2),[1,size(x,2)]);  %product for this configuration
denom = q{nnvars}(idx);
denom(find(denom==0)) = realmin;
result(idx) = result(idx) + pp./denom;
end
% update global distribution
a = repmat(iir{nnvars},[1,1,qq]);
b = repmat(jjr{nnvars},[1,1,qq]);
c = permute(repmat((1:qq)',[1 size(a,1) size(a,2)]),[2 3 1]);
gidx = sub2ind(size(buffer),a,b,c);
buffer(gidx) = result;
end
% initialize r from the global data in buffer
for nnfuns = minfuns:maxfuns
    a = repmat(iic{nnfuns},[1,1,qq]);
b = repmat(jjc{nnfuns},[1,1,qq]);
c = permute(repmat((1:qq)',[1 size(a,1) size(a,2)]),[2 3 1]);
gidx = sub2ind(size(buffer),a,b,c);
    r{nnfuns} = buffer(gidx);
end
% vertical step
buffer = zeros([size(H),qq]);
QQ = zeros(qq,size(H,2));
for nnfuns = minfuns:maxfuns
    if(~isempty(Hc{nnfuns}))  % make sure the are variables for this case
        % calculate products
        pp = repmat( prod ( r{nnfuns},1),[size(r{nnfuns},1),1]).*ff{nnfuns};  % product for this configuration
        denom = r{nnfuns};
        denom(find(denom==0)) = realmin;
        result = pp./denom;
        result = result./repmat((sum(result,3)),[1,1,qq]);  % normalize to distribution
        % update global distribution
        a = repmat(iic{nnfuns},[1,1,qq]);
b = repmat(jjc{nnfuns},[1,1,qq]);
c = permute(repmat((1:qq)',[1 size(a,1) size(a,2)]),[2 3 1]);
gidx = sub2ind(size(buffer),a,b,c);
        buffer(gidx) = result;
        Q{nnfuns} = pp.*ff{nnfuns};
b = repmat(jjc{nnfuns}(1,:),[qq,1]);
c = repmat((1:qq)',[1, size(b,2)]);
qidx = sub2ind(size(QQ),c,b);
QQ(qidx) = squeeze(Q{nnfuns}(1,:,:))';
end
end
% initialize q from the global data in buffer
for nnvars = minvars:maxvars
    a = repmat(iir{nnvars},[1,1,qq]);
b = repmat(jjr{nnvars},[1,1,qq]);
c = permute(repmat((1:qq)',[1 size(a,1) size(a,2)]),[2 3 1]);
gidx = sub2ind(size(buffer),a,b,c);
    q{nnvars} = buffer(gidx);
end
% tentative decoding

QQ = QQ ./ repmat(sum(QQ,1),[qq 1]); % normalize - can be used as soft outputs
[xi xj sx] = find(QQ == repmat(max(QQ),[size(QQ,1),1]));
x_hat = xi-1;
end
end