Technical University of Crete

School of Electronic and Computer Engineering

Low-Density Parity-Check Codes for the relay channel

by

Panagiotis Chatzinikolaou

Submitted on July 29, 2014 in partial fulfilment of the requirements for the Electronic and Computer Engineering diploma degree.

Abstract

Since 1948, when Claude Shannon introduced the notion of channel capacity, the ultimate goal of coding theory has been to find practical capacity-approaching codes. This thesis is concerned with the analysis, design, construction, and, mainly, the decoding of an extremely powerful and flexible family of error-control codes, called low-density parity-check (LDPC) codes. LDPC codes can be designed to perform close to the capacity of many different types of channels with a practical decoding complexity. The main design tool, Density Evolution, which predicts the asymptotic performance of a belief-propagation decoder is analysed. An effective construction method, called configuration model, for picking LDPC codes at random from LDPC ensembles is presented in depth. In addition, a construction tool for LDPC codes of moderate lengths, namely the Progressive Edge Growth algorithm, which tries to maximize the girth of a code is presented. Furthermore, we present the analysis,design, construction and decoding of bilayer LDPC codes for the half-duplex relay channel. To analyse the performance of bilayer LDPC codes, bilayer Density Evolution is presented as an extension of the standard Density Evolution. Finally, we analyse in depth the construction procedure of bilayer LDPC codes as an extension of single user's configuration model.

Acknowledgements

First of all, I would like to thank my parents for their endless support, love and patience without whom this thesis may not have been completed. Many thanks go to my thesis supervisor, Professor Athanasios Liavas, for all his guidance, patience and for the necessary contributions to this thesis. I was very lucky to have a supervisor who cared so much about my thesis, and who always responded to my questions. Also, I would like to thank my thesis committee for the knowledge they provided me during my studies, and for the time devoted to read this thesis. Last but not least, I would like to thank all my friends and colleagues for the memorable moments we had.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

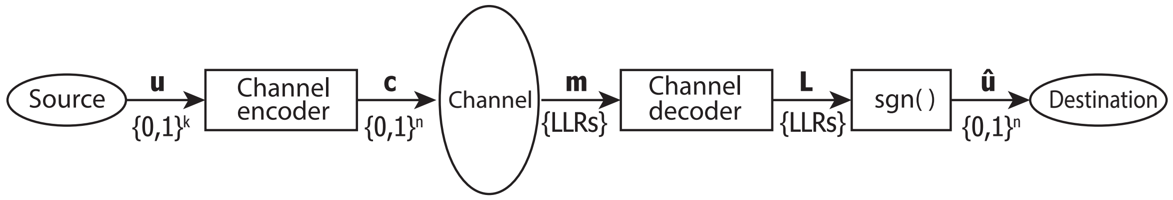


Figure 1.1: A typical communications system.

Suppose we have a noisy communication channel through which we wish to send information reliably. Shannon [2] showed that arbitrarily reliable transmission is possible through this channel if the information rate, in bits per channel use, is less than the *channel capacity* of the channel. He proved that it is possible to transmit digital data with arbitrarily high reliability, over noise-corrupted channels, by encoding the digital message with an *error correction code* prior to transmission and subsequently decoding it at the receiver. The error correction encoder maps each vector $\mathbf{u}$ of k digits, representing the message, to longer vectors $\mathbf{c}$ of n digits, known as codewords. The redundancy implicit in the transmission of codewords, rather than the raw data alone, is the quid pro quo for achieving reliable communication over intrinsically unreliable channels. The code rate $r = \frac{k}{n}$ determines the amount of redundancy added by the error correction code. The transmitted symbols may be corrupted in some way by the channel, and it is the function of the error correction decoder to use the added redundancy to determine the corresponding k message bits despite the imperfect reception. **Fig. 1.1** shows a typical communications system.

1.2 Brief history of LDPC codes

LDPC codes were invented by Robert Gallager in 1960 [4] but forgotten for over 30 years, because they were considered too complex at the time of their discovery. Parallel to the research on turbo codes and influenced by the focus on turbo codes, in 1996 MacKay and Neal in [21] and Sipser and Spielman in [22] rediscovered this long forgotten class of codes.

In 2001, LDPC codes drew a lot of attention as they had very good performance. As shown in [6, 11], irregular LDPC codes can significantly outperform regular LDPC codes. All LDPC codes which can approach the Shannon limit are irregular codes. Another feature of LDPC codes is their simple graphical representation [5] which leads to accurate asymptotic analysis [6].

LDPC codes are already used in some standards such as ETSI EN 302 307 for digital video broadcasting and 802.16 (Broadband Wireless Access Working Group) for coding on orthogonal frequency division multiple access (OFDMA) systems [25].

Chapter 2

Channels, codes and capacity

2.1 Binary input symmetric output memoryless channels

A *discrete channel* has input a symbol X from a discrete alphabet $\mathcal{X}$, known as the source alphabet, and has output a symbol Y from a possibly different discrete alphabet, $\mathcal{Y}$. A binary input channel transmits two discrete symbols, usually 0,1 or modulated +1, −1. Unfortunately, the channels do not always map a given transmitted symbol to the same received symbol (which is why we need error correction).

A communications channel can be modelled as follows. For a given symbol x_i transmitted at time i, such that $x_i \in \mathcal{X}$, the channel transition probability $p(y/x) = p(Y = y_i/X = x_i)$ gives the probability that the returned symbol Y at time i is the symbol $y_i \in \mathcal{Y}$. A channel is called *memoryless* if the channel output at any time instant depends only on the input at that time instant. This means that for a sequence of transmitted symbols $\mathbf{x} = [x_1, x_2, \cdots, x_N]$ and received symbols $\mathbf{y} = [y_1, y_2, \cdots, y_N]$:

$$p(\mathbf{y}/\mathbf{x}) = \prod_{i=1}^{N} p(y_i/x_i). \qquad \textbf{2.1}$$

Hence, a memoryless channel is completely described by its input and output alphabets and the conditional probability distribution $p_{Y/X}(y/x)$ for each input-output symbol pair.

The three channels we consider in this thesis are the binary symmetric channel (BSC), the binary erasure channel (BEC), and the binary input additive white Gaussian noise (BI-AWGN) channel. They are all binary input memoryless channels.
A binary input channel is *symmetric* if both input bits are corrupted equally by the channel. We define *binary-input symmetric-output memoryless* channels as:

Definition 2.1. A binary-input symmetric-output memoryless channel is a $discrete-time$ channel whose input X is ± 1 or $\{0,1\}$, and output Y (*discrete* or *continuous*) depends only on the current input symbol and satisfies the following:

$$p(Y = y/X = 1) = p(Y = -y/X = -1). \qquad \textbf{2.2}$$

$\diamond$

Gallager [**?**] proved that the *channel capacity* of a *symmetric* channel can be achieved using equi-probable inputs. Therefore, the capacity-achieving input distribution of a binary-input output-symmetric channel is uniform.

2.1.1 Binary Symmetric Channel (BSC)

The *binary symmetric channel*, shown in **Fig. 2.1**, transmits one of two symbols, the binary digits $X \in \{0,1\}$ and returns one of two symbols, $Y \in \{0,1\}$. The channel flips a transmitted bit with probability ϵ and with probability $1-\epsilon$ the symbol y is the symbol that was sent. The parameter ϵ is called the *crossover probability* of the channel.

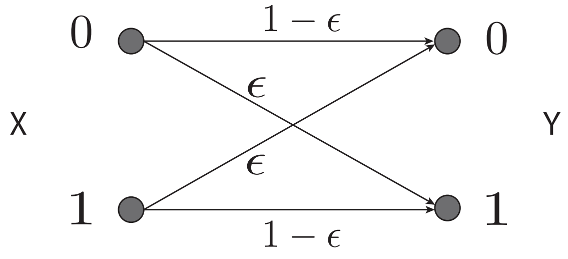


Figure 2.1: Binary Symmetric Channel (BSC).

So, the transition probabilities for the BSC are:

$$\begin{aligned} p(Y = 0/X = 0) &= 1 - \epsilon \\ p(Y = 0/X = 1) &= \epsilon \\ p(Y = 1/X = 0) &= \epsilon \\ p(Y = 1/X = 1) &= 1 - \epsilon. \end{aligned}$$

As stated previously, a binary input channel is *symmetric* if both input symbols are corrupted equally by the channel. The BSC channel is *symmetric* since $p(Y = 0/X = 1) = p(Y = 1/X = 0)$ and $p(Y = 0/X = 0) = p(Y = 1/X = 1)$. At the decoder, the output Y received from the channel is used to decode the symbol X that was sent. In this case, we are interested in the (a-posteriori) probability $p(x/y)$. Assuming that the input symbols are equally likely, the log-likelihood ratios (LLRs) for the i-th transmitted bit are:

$$LLR_i = \ell(x_i/y_i) \triangleq \log \frac{p(X_i = 0/y_i)}{p(X_i = 1/y_i)} \overset{\text{bayes}}{=} \log \frac{p(y_i/X_i = 0)}{p(y_i/X_i = 1)} = \begin{cases} \log \frac{\epsilon}{1-\epsilon}, & \text{if } y_i = 1; \\ \log \frac{1-\epsilon}{\epsilon}, & \text{if } y_i = 0. \end{cases} \quad \textbf{2.3}$$

It can be proved that the capacity of BSC is:

$$C(\epsilon) = 1 - H_2(\epsilon), \quad \textbf{2.4}$$

where $H_2(\epsilon) = -\epsilon \log_2(\epsilon) - (1 - \epsilon) \log_2(1 - \epsilon)$ is the binary entropy function.

2.1.2 Binary Erasure Channel (BEC)

The *binary erasure channel* shown in **Fig. 2.2**, transmits one of two symbols, usually the binary digits $X \in \{0, 1\}$. However, the receiver either receives the bit correctly or it receives a message "?" that the bit was erased. In other words, BEC does not introduce errors. The BEC erases a bit with erasure probability ϵ. Hence, the channel transition probabilities are:

$$p(Y=0/X=0)=1-\epsilon.$$
$$p(Y=0/X=1)=0.$$
$$p(Y=1/X=1)=1-\epsilon.$$
$$p(Y=1/X=0)=0.$$
$$p(Y=?/X=0)=\epsilon.$$
$$p(Y=?/X=1)=\epsilon.$$

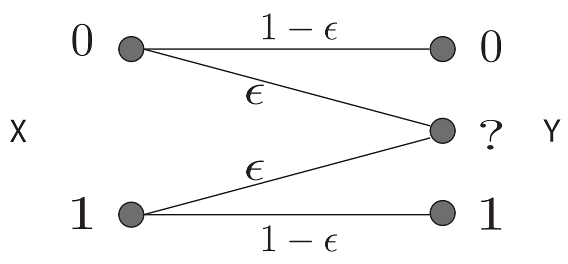


Figure 2.2: Binary Erasure Channel (BEC).

We observe that the BEC is *symmetric*. The BEC does not flip bits, therefore, if Y is received as 1 or 0 then the receiver knows the value of X. On the other hand, if the channel has erased the transmitted bit, the receiver has no information about X and can only use the a priori probabilities of the source. If the source is equi-probable (i.e. the input bits 1 and 0 are equally likely) the receiver can only make a fifty-fifty guess:

$$p(X=0/Y=?)=0.5$$
$$p(X=1/Y=?)=0.5.$$

Hence, for this channel, we have that the *received* LLRs for the i-th transmitted bit are

$$LLR_i=\ell(x_i/y_i)\triangleq\log\frac{p(X_i=0/y_i)}{p(X_i=1/y_i)}\overset{\text{bayes}}{=}\log\frac{p(y_i/X_i=0)}{p(y_i/X_i=1)}=\begin{cases}\log\frac{0}{1}=-\infty, & \text{if } y_i=1,\\ \log\frac{\epsilon}{\epsilon}=0, & \text{if } y_i=?,\\ \log\frac{1}{0}=\infty, & \text{if } y_i=0.\end{cases} \quad \textbf{2.5}$$

It can be proved that the capacity of BEC is:

$$C(\epsilon)=1-\epsilon. \quad \textbf{2.6}$$

2.1.3 Binary AWGN Channel (BI-AWGN)

The last channel we consider, is a binary input channel with additive noise modelled as white Gaussian. The BI-AWGN channel shown in **Fig. 2.3** is described by the equation:

$$Y = X + N,\ X \in \{1, -1\} \text{ and } N \sim \mathcal{N}(0, \sigma^2).$$

From the channel model, it can be inferred that output Y, conditioned on the input, follows the Gaussian distribution with mean either $+1$ or -1 and variance σ^2.

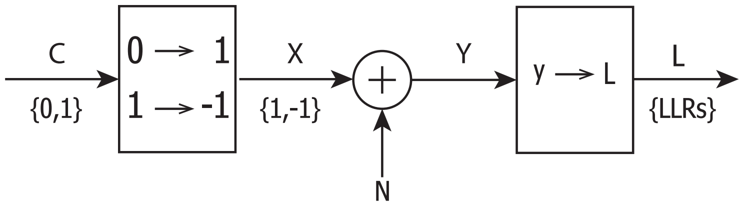


Figure 2.3: The BI-AGWN channel.

Assuming equi-probable transmitted symbols X_i, the *received* LLRs are:

$$\begin{aligned}
\ell_i = \ell(x_i/y_i) &\triangleq \frac{p(C_i = 0/y_i)}{p(C_i = 1/y_i)} \\
&= \frac{p(X_i = 1/y_i)}{p(X_i = -1/y_i)} \\
&\overset{\text{bayes}}{=} \frac{p(y_i/X_i = 1)}{p(y_i/X_i = -1)} \\
&= \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-1)^2}{2\sigma^2}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i+1)^2}{2\sigma^2}\right)} \\
&= \frac{2y_i}{\sigma^2} \sim \mathcal{N}\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right), \text{ assuming } y_i \sim \mathcal{N}(1, \sigma^2).
\end{aligned} \qquad \textbf{2.7}$$

The LLR value of a bit C_i is sometimes called a *soft decision* for C_i. A *hard decision* for C_i or, equivalently, for X_i will be:

$$\hat{X}_i = \begin{cases} +1, & \text{if } \ell_i > 0, \\ -1, & \text{if } \ell_i < 0. \end{cases}$$

We can characterise the channel by $\frac{E_N}{\sigma^2}$ the ratio of the energy per transmitted bit to the noise energy σ^2. For our setting $E_N = 1$ since $X \in \pm 1$. The measure $\frac{E_b}{N_0}$ is often quoted where E_b is the energy per information bit, $E_b = \frac{E_N}{r}$, r is the rate, and $N_0 = 2\sigma^2$ is the double-sided power spectral density. We therefore have $\frac{E_b}{N_0} = \frac{E_N}{2r\sigma^2}$.

Finally, we can observe that **2.2** holds, hence, BI-AWGN channel is *symmetric*. The log-likelihood ratio (LLR) ℓ as a fraction of the channel output y is defined as

$$\ell(y) \triangleq \log \frac{p(y/X=1)}{p(y/X=-1)}. \qquad \textbf{2.8}$$

Lemma 2.1. From the channel symmetry condition it follows [6] that

$$\ell(y) = -\ell(-y). \qquad \textbf{2.9}$$

Proof.

$$\ell(y) = \log \frac{p(y/X=1)}{p(y/X=-1)} = \frac{p(-y/X=-1)}{p(-y/X=1)} = -\ell(-y)$$

. ■

We call ℓ the *initial message* of the channel and the distribution $p(\ell)$ of ℓ the *initial density* of the channel, which will be used later for *density evolution* and *Gaussian approximation*. Richardson et al. [6] showed that initial densities satisfy the following *symmetry condition*:

$$p(\ell) = e^{\ell} p(-\ell). \qquad \textbf{2.10}$$

Densities of initial LLRs for BEC, BSC and BI-AWGN channels satisfy the aforementioned condition. As we will see in Chapter 5, $p(\ell)$ is the so called $\mathcal{L}$-density. Two important facts that we will use are: (i) $\mathcal{L}$-distributions for binary memoryless symmetric-output channels are always symmetric and (ii) the convolution of symmetric distributions is symmetric [6].

For the BI-AWGN channel a calculation for the capacity (assuming equi-probable source) is as follows:

$$\begin{aligned} C_{\mathrm{BIAWGN}}(\sigma) &= I(X;Y) \\ &= H(Y) - H(Y/X) \\ &= H(Y) - H(Z),\ Z \sim \mathcal{N}(0,\sigma^2) \end{aligned}$$

It can be proved that $H(Z) = \frac{1}{2}\log_2 2\pi e\sigma^2$. Using total probability theorem, $p(y) = \frac{1}{2}p(y/X=1) + \frac{1}{2}p(y/X=-1)$ where

$$p(y/X=\pm 1) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y\pm 1)^2}{2\sigma^2}\right).$$

Thus,

$$p(y) = \frac{1}{\sqrt{8\pi\sigma^2}}\left(\exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) + \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right)\right).$$

Therefore,

$$C_{\mathrm{BIAWGN}}(\sigma) = -\int_{-\infty}^{\infty} p(y)\log_2 p(y)\ dy - \frac{1}{2}\log_2 2\pi e\sigma^2. \qquad \mathbf{2.11}$$

For a channel with noise level parameter x and an error correction code with rate r, the noise level x_{sh}, such that $C(x_{\mathrm{sh}}) = r$, is a threshold for error correction codes with that rate. The noise level x_{sh} is called the *Shannon limit.* Shannon's noisy coding theorem says that for any noise level x below x_{sh} there exists a code with rate r that can achieve an arbitrarily low probability of error, while for any noise level above x_{sh}, no rate-r code can achieve an arbitrarily low probability of error.

To find the Shannon limit of error correction coding with rate r, and channel parameter x, requires that we find a value for x_{sh} such that $C(x_{\mathrm{sh}}) = r$. In the case of BI-AWGN channel, channel parameter x, is referred to standard deviation σ. Pseudo-code for computing the Shannon limit on a BI-AWGN channel within a tolerance δ is given in algorithm below [13, p.15]. The symbol r represents the rate of the code and σ_H, σ_L are the upper and lower limits in order to search for σ_{sh}.

Algorithm 1 Shannon Limit of a BI-AWGN channel

```
1: procedure ShannonLimit($r$,$\delta$,$\sigma_L$,$\sigma_H$)
2:   repeat
3:     $\sigma = \frac{1}{2}(\sigma_L + \sigma_H)$
4:     $C_{\text{BIAWGN}}(\sigma) = \int_{-\infty}^{\infty} p(y) \log_2 p(y)\ dy$
5:     $C_{\text{BIAWGN}}(\sigma) = C_{\text{BIAWGN}}(\sigma) - \frac{1}{2} \log_2 2\pi e \sigma^2$
6:     if $C_{\text{BIAWGN}}(\sigma) > r$ then
7:       $\sigma_L = \sigma$
8:     else
9:       $\sigma_H = \sigma$
10:    end if
11:  until $\sigma_H - \sigma_L < \delta$
12:    $\frac{E_b}{N_0} = 10 \log_{10} \frac{1}{2r\sigma^2}$
13:    return $\frac{E_b}{N_0}$
14: end procedure
```

2.2 Linear Block Codes

Assume that the output of an information source is a continuous sequence of binary symbols over GF(2)=$\{0,1\}$, called an *information sequence.* The binary symbols in an information sequence are called information bits. In block coding, an information sequence is segmented into message blocks of fixed length; each message block consists of k information bits. There are 2^k distinct messages. At the channel encoder, each input message $\mathbf{u}$ of k information bits is encoded into a longer binary sequence $\mathbf{c}$ of n binary digits with $n > k$, according to certain encoding rules. This longer sequence is called the *codeword* of message $\mathbf{u}$. Since there are 2^k distinct messages, there are 2^k distinct codewords one for each distinct message. This set of 2^k codewords, denoted by $\mathcal{C}$, is said to form an (n,k) block code. For a block code to be useful, the 2^k codewords must be distinct. The $n-k$ bits added to each input message by the channel encoder are called *redundant bits.* The redundant bits carry no new information and their main function is to provide the code with the capability of *detecting* and *correcting* transmission errors caused by the channel noise or interferences. How to form these redundant bits such that an (n,k) block code has good error-correcting capability is a major concern in designing the channel encoder.

For a block code of length n with 2^k codewords, unless it has certain special structure properties, the encoding and decoding scheme would be complex for large k since the encoder has to store 2^k codewords of length n and the decoder has to perform a table (with 2^n entries) look-up to determine the transmitted codeword [20, p.95]. Therefore, we must restrict our attention to block codes that can be implemented in a practical manner. A desirable structure for a block code is *linearity*.

Definition 2.2. A binary block code of length n and 2^k codewords is said to be a $\mathcal{C}(n,k)$ linear code if, and only if, its 2^k codewords form a k-dimensional subspace of the vector space of the binary n-tuples over GF(2).

◇

The case $k = 0$ corresponds to the trivial linear code which consists only of the all-zero codeword $\mathbf{0}$. Since the code forms a subspace it contains the all-zero codeword.

In other words, a code $\mathcal{C}$ over a field $\mathbb{F}$ is *linear* if it is closed under n-tuple addition and scalar multiplication:

$$\alpha_1 \cdot \mathbf{c}_1 + \alpha_2 \cdot \mathbf{c}_2 \in \mathcal{C}, \quad \forall \mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C} \text{ and } \forall \alpha_1, \alpha_2 \in \mathbb{F}. \tag{2.12}$$

Definition 2.3. The rate of a code $\mathcal{C}(n,k)$ is defined as

$$r \triangleq \frac{k}{n} \text{ information bits/codeword.}$$

◇

Definition 2.4. ([23, p.83]). The *Hamming weight* of a codeword $\mathbf{c}$ is the number of non-zero components of the codeword. The minimum weight of a linear code $\mathcal{C}$, is the smallest Hamming weight of any non-zero codeword. ◇

Definition 2.5. ([23, p.83]). The minimum-distance of a linear code $\mathcal{C}$, is equal to the minimum weight of any non-zero codeword. ◇

Recall that, since a linear code $\mathcal{C}$ forms a subspace of dimension k, there exist k linear independent binary vectors in $\{0,1\}^n$, which form a basis of this subspace. We denote these k vectors as $\mathbf{g}_0, \mathbf{g}_2, \cdots, \mathbf{g}_{k-1}$. Consequently, any linear combination of them generates a codeword in $\mathcal{C}$.

Definition 2.6. A generator matrix $\mathbf{G}$ for a (n,k) linear code $\mathcal{C}$ (over field $\mathbb{F}_2 = \{0,1\}$) is a k-by-n matrix whose row space is the given code. In other words, $\mathcal{C} = \{\mathbf{x}\mathbf{G} \mid \mathbf{x} \in \mathbb{F}_2^k\}$. ◇

Obviously, the rank of a generator matrix $\mathbf{G}$ of a linear code $\mathcal{C}$ over $\mathbb{F}_2$ equals the dimension of $\mathcal{C}$.

$$\mathbf{G} \triangleq \begin{bmatrix} \mathbf{g}_0 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}. \qquad \mathbf{2.13}$$

Let $\mathbf{u}$ be an information vector. The encoding process is as follows:

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \text{ where } \mathbf{c} \in \mathcal{C}.$$

Therefore, the codeword $\mathbf{c}$ for a message $\mathbf{u}$ is simply a linear combination of the rows of matrix $\mathbf{G}$ with the information bits in the message $\mathbf{u}$ being the coefficients.
The utilization of generator matrix $\mathbf{G}$ is preferable, since it costs less storage.

Note that the representation of the code provided by $\mathbf{G}$ is not unique. From a given generator matrix $\mathbf{G}$, another generator $\mathbf{G}'$ can be obtained by performing row operations. Then an encoding operation defined by $\mathbf{c} = \mathbf{u}\mathbf{G}'$ maps the message $\mathbf{u}$ to a codeword in $\mathcal{C}$, but it is not necessarily the same codeword that would be obtained using the generator $\mathbf{G}$.

Example 2.1. A generator matrix for the $(7,4)$ Hamming code is

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Encoding the message $\mathbf{u} = [1\,0\,0\,1]$, we have

$$\mathbf{c} = \mathbf{u}\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Another generator $\mathbf{G}'$ is obtained by replacing the first row of $\mathbf{G}$ with the sum of the first two rows of $\mathbf{G}$, hence,

$$\mathbf{G}' = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

For $\mathbf{u}$ the corresponding codeword using $\mathbf{G}'$ is

$$\mathbf{c}' = \mathbf{u}\mathbf{G}' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \neq \mathbf{c},$$

where $\mathbf{c}' \in \mathcal{C}$. □

Definition 2.7. A linear block code $\mathcal{C}$(n,k) is called systematic if its k-by-n generator matrix $\mathbf{G}$ has the form

$$\left(\mathbf{I}_k \mid \mathbf{P}\right) \qquad \mathbf{2.14}$$

where $\mathbf{I}_k$ denotes the k-by-k identity matrix and matrix $\mathbf{P}$ is a k-by-$(n-k)$ matrix. ◇

The generator matrix $\mathbf{G}$ of a linear block (n, k, d_{min}) code $\mathcal{C}$ can be brought to a systematic form by elementary row operations and/or column permutations. Performing elementary row operations (replacing a row with linear combinations of some rows) does not change the row span, so that the same code is produced. If two columns of a generator matrix are interchanged, then the corresponding positions of the code are changed, but the distance structure of the code is preserved.
When using a systematic generator matrix $\mathbf{G} = (\mathbf{I}_k \mid \mathbf{P})$ for encoding, the mapping

$$\mathbf{u} \longmapsto \mathbf{u}\mathbf{G} \qquad \mathbf{2.15}$$

takes the form

$$\mathbf{u} \longmapsto \left(\mathbf{u} \mid \mathbf{u}\mathbf{P}\right) \qquad \mathbf{2.16}$$

where information word $\mathbf{u} \in \mathbb{F}_2^k$. That is, the first k entries in the encoded codeword form the information word; which is very useful for decoding. The second part $\mathbf{uP}$, consists of parity check symbols.

Definition 2.8. Let $\mathcal{C}$ be an (n, k) linear code over $\mathbb{F}_2$. A matrix $\mathcal{H}$ with the property that $\mathcal{H}\mathbf{x}^T = \mathbf{0}$ if, and only if, $\mathbf{x} \in \mathcal{C}$ is called a parity-check matrix for $\mathcal{C}$. ◇

In other words, the parity check matrix is a $(n-k)$-by-n matrix, whose rows are orthogonal to the row space of matrix $\mathbf{G}$, i.e., all rows of $\mathcal{H}$ belong to the nullspace of $\mathbf{G}$. That is, $\mathbf{G}\mathcal{H}^T = \mathbf{0}_{k\times(n-k)}$.

In the special case where $\mathbf{G}$ is a systematic matrix, according to 2.14, we can take the $(n-k) \times k$ matrix $\mathcal{H} = (\mathbf{P}^T \mid \mathbf{I}_{n-k})$ as a parity-check matrix. Notice that, if we did not restrict ourselves to binary linear block codes, the parity-check matrix $\mathcal{H}$ would have the form $(-\mathbf{P}^T \mid \mathbf{I}_{n-k})$.

Corollary 2.1. Any codeword $\mathbf{c}$ belongs to $\mathcal{C}$ if, and only if, it is perpendicular in any row of parity check matrix, namely

$$\mathbf{c} \in \mathcal{C} \Longleftrightarrow \mathbf{c}\mathcal{H}^T = \mathbf{0}. \qquad \mathbf{2.17}$$

That is, the codewords in $\mathcal{C}$ lie in the (left) nullspace of $\mathcal{H}$. The condition $\mathbf{c}\mathcal{H}^T = \mathbf{0}$ imposes linear constraints among the bits of $\mathbf{c}$ called the *parity-check equations*.

Theorem 2.1. ([23], p.30). A linear block code with minimum distance d can correct at most $\lfloor \frac{d-1}{2} \rfloor$ errors using minimum-distance decoding. This bound is called the error correcting capability of the code.

Definition 2.9. Let $\mathcal{C}(n,k)$ be a linear code. The *dual code* of $\mathcal{C}$, denoted by $\mathcal{C}^{\perp}(n, n-k)$, consists of all vectors $\mathbf{x} \in \mathbb{F}^n$ such that $\mathbf{x}\mathbf{c}^T = 0$ for all $\mathbf{c} \in \mathcal{C}$. That is, the codewords of $\mathcal{C}^{\perp}$ are "orthogonal" to $\mathcal{C}$. ◇

An equivalent definition of a *dual code* is given by

$$\mathcal{C}^{\perp} = \{\mathbf{x} \in \mathbb{F}^n : \mathbf{x}\mathbf{G}^T = \mathbf{0}\}. \qquad \mathbf{2.18}$$

Definition 2.10. Let $\mathcal{C}(n,k)$ be a linear code over $\mathbb{F}_2 = GF(2)$. Consider a received codeword $\mathbf{y}$. The vector $\mathbf{s} = \mathcal{H}\mathbf{y}^T$ is called the *syndrome* of the received word. The vector $\mathbf{y}$ is a codeword if, and only if, its *syndrome* is equal to 0. ◇

Example 2.2. Consider a $(7,4)$ Hamming code generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \qquad \mathbf{2.19}$$

Note that matrix $\mathbf{G}$ is a systematic generator matrix, since $\mathbf{G} = (\mathbf{I}_4 \mid \mathbf{P}_{4\times 3})$. The rate r of the code is $\frac{4}{7}$. The minimum distance of the code (i.e. the minimum weight of any codeword) is 3.

There are 2^4 codewords:

$$\begin{aligned} \mathcal{C} = \{&(0000000), (0001111), (0010110), (0011001), (0100101), \\ &(0101010), (0110011), (0111100), (1000011), (1001100), \\ &(1010101), (1011010), (1100110), (1101001), (1110000), (1111111)\} \end{aligned}$$

The systematic parity check matrix is of the form

$$\mathcal{H} = (\mathbf{P}^T_{4\times 3} \mid \mathbf{I}_3),$$

and is given by

$$\mathcal{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Now let us consider an information word, e.g., $\mathbf{u} = [\ 0\ 1\ 0\ 0\ ]$. According to **2.15**, we take the codeword

$$\mathbf{c} = \mathbf{uG} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}. \qquad \mathbf{2.20}$$

That is, the first 4 bits describe the information part of the codeword and the last 3 bits describe the redundant part. In addition, if we observe the codewords of $\mathcal{C}$, we will notice that the codeword $\mathbf{c}$ belongs to $\mathcal{C}$. Let us verify the aforementioned claim, i.e.,

$$\mathcal{H}\mathbf{c}^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \qquad \mathbf{2.21}$$

□

Chapter 3

Low-Density Parity-Check (LDPC) codes

LDPC codes constitute a powerful class of linear block codes which provide near Shannon-limit performance on a large variety of channels while simultaneously admitting implementable decoders. The difference between an LDPC code and conventional linear codes is the fact that the parity-check matrix of an LDPC code is sparse, i.e., the number of non-zero entries is much smaller than the total number of entries. Since an LDPC code belongs to the class of linear block codes, it can be represented by a bipartite graph called *Tanner graph* [14].

3.1 Matrix representation of LDPC codes

We consider *only* binary LDPC codes and we denote with $\mathbb{F}_2$ the GF(2). Since LDPC codes form a class of linear block codes, they can be described as a certain k-dimensional subspace $\mathcal{C}$ of the vector space $\mathbb{F}_2^n$ of binary n-tuples over $\mathbb{F}_2$. Given this, we can find a basis $\mathbf{B} = \{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}\}$ which spans $\mathcal{C}$ so that each $\mathbf{c} \in \mathcal{C}$ may be written as $\mathbf{c} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}$ for some $\{u_i\}_{i=0}^{k-1} \in \mathbb{F}_2$. Equivalently, the aforementioned linear combination can be rewritten as $\mathbf{c} = \mathbf{uG}$, where $\mathbf{u} = [u_0\, u_1\, \ldots\, u_{k-1}]$ and $\mathbf{G}$ is the generator matrix whose rows are the (row) vectors $\{\mathbf{g}_i\}$. Since $\mathbf{G}$ is a basis, $\{\mathbf{g}_i\}$ are linearly independent, hence, matrix $\mathbf{G}$ has full row rank. From 2.18, it follows that the $(n-k)$ dimensional null (or dual) space $\mathcal{C}^{\perp}$ of $\mathbf{G}$ is spanned by the basis $\mathbf{B}^{\perp} = \{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}\}$. Thus, for each $\mathbf{c} \in \mathcal{C}$, $\mathbf{ch}_i^T = 0$, for $i = 0, \ldots, n-k-1$, or, more compactly, $\mathbf{c}\mathcal{H}^T = \mathbf{0}$, where $\mathcal{H}_{(n-k)\times n}$ is the so called parity-check matrix whose rows are the vectors $\{\mathbf{h}_i\}$, and is the generator for the dual space $\mathcal{C}^{\perp}$. The parity-check matrix $\mathcal{H}$ is so named, because it performs $(n-k)$

separate parity checks on a received word.

LDPC codes can be classified into two types: *regular* and *irregular*. **Regular** LDPC codes have a constant number of symbols participating in each parity-check equation, and each symbol participates in a constant number of parity-check equations. That is, the column and row weights (i.e. number of 1s) of the parity-check matrix are constant as shown in **3.1**. They were originally proposed by Gallager [4] in 1962 who proved that they are *asymptotically* good in the sense that their *minimum distance* grows *linearly* with block length. This guarantees that with ML decoding, the codes do not suffer from the error floor phenomenon, a flattening of the bit error rate (BER) curve that results in poor performance at high signal-to-noise ratios (SNRs). Similar behaviour is observed with iterative BP decoding as well. However, the iterative decoding behaviour of regular codes in the so-called *waterfall*, or moderate BER, region of the performance curve falls short of capacity, making them unsuitable for severely power-constrained applications, such as uplink cellular data transmission or digital satellite broadcasting systems, that must achieve the best possible performance at moderate BERs [24].

On the other hand, **irregular** LDPC codes, pioneered by Luby et al. [8] in 2001, have not constant column and row weights. In fact, column and row weights are chosen according to some distribution. Thus, an irregular LDPC code might have a matrix representation in which half rows of the parity-check matrix have weight 3 and half have degree 5, while half columns have weight 6 and half have weight 8. In the case where *all* columns have the same weight, then we name this code *concentrated irregular* LDPC code. In addition, irregular codes exhibit capacity approaching performance in the waterfall but are normally subject to an error floor, making them undesirable in applications, such as data storage and optical communication, that require very low decoded BERs [24]. Typical performance characteristics of regular and irregular LDPC codes on an additive white Gaussian noise (AWGN) channel are illustrated in **Fig. 3.1** where the SNR is expressed in terms of $\frac{E_b}{N_0}$, the *information bit signal-to-noise ratio.*

Example 3.1. ((d_{v}, d_c)-Regular LDPC code). Consider the following parity-check matrix $\mathcal{H}_{5\times 10}$ where d_{v} denotes the number of ones in each column of $\mathcal{H}$ and d_c denotes the number of ones in each row. Hence, for our corresponding matrix we have d_{v}=3 and d_c=6. The rows of $\mathcal{H}$ represent the parity-check equations and columns represent the codeword bits. Also, we define as m the number of parity-check equations, i.e., $m \triangleq n-k$. As we will see in the next subsection all variable nodes will have degree 3 and all check nodes will have degree 6.

$$\mathcal{H} = \begin{array}{c|cccccccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ c_1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ c_2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ c_3 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ c_4 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ c_5 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}. \qquad \mathbf{3.1}$$

The *code rate* r is

$$r = \frac{k}{n} \geq \frac{n-m}{n} = 1 - \frac{d_{\mathrm{v}}}{d_c}. \qquad \mathbf{3.2}$$

If the rows of $\mathcal{H}$ are *linearly independent*, i.e., rank($\mathcal{H}$)$=n-k$, then $r = 1 - \frac{d_{\mathrm{v}}}{d_c}$. The quantity $\frac{n-m}{n}$ is referred to as the *design rate* [6].
Moreover, each parity-check equation c_i, for $i = 1, \ldots, 5$, is an *even parity* constraint on its codeword bits, i.e.,

$$\begin{aligned} c_1 &: x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8 = 0 \\ c_2 &: x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_9 \oplus x_{10} = 0 \\ c_3 &: x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8 = 0 \\ c_4 &: x_2 \oplus x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{10} = 0 \\ c_5 &: x_1 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_9 \oplus x_{10} = 0. \end{aligned}$$

□

Example 3.2. (Irregular LDPC code). Consider the parity-check matrix

$$\mathcal{H}' = \begin{array}{c|cccccccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ c_1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ c_2 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ c_3 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ c_4 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ c_5 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}, \qquad \mathbf{3.3}$$

which corresponds to an irregular LDPC code. Observe that in contrast to matrix $\mathcal{H}$, rows and columns do not have constant number of ones. The *design rate* is calculated in a more complex manner in contrast to the regular case, hence, we omit it for now. Parity-check equations can be derived as in the regular case.

$\square$

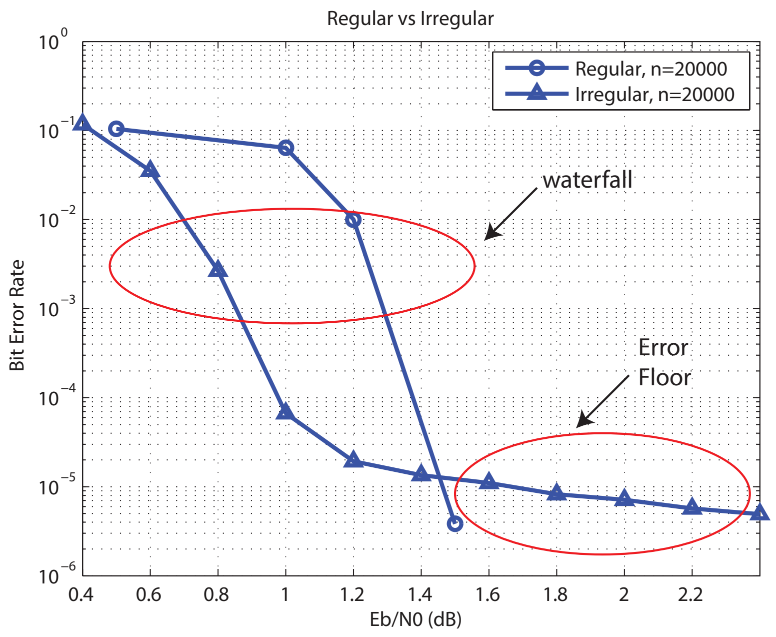


Figure 3.1: Bit Error Rate of a $(3,6)-$*regular* and a *concentrated irregular*[1] LDPC code with same block length over BI-AWGN channel. We used BPSK modulation.

[1]The degree distributions are:

$$\begin{aligned}\lambda(x) &= 0.169010x + 0.161244x^2 + 0.005938x^4 + 0.016799x^5 + 0.186455x^6\\ &+ 0.06864x^{13} + 0.025890x^{16} + 0.096393x^{18} + 0.010531x^{26}\\ &+ 0.004678x^{27} + 0.079616x^{28} + 0.011885x^{38} + 0.224691x^{99}\\ \textit{and}\ \ \rho(x) &= x^{10}.\end{aligned}$$

We'll explain the meaning of these polynomials after next subsection.

3.2 Graphical representation of LDPC codes

LDPC codes are linear codes that can be described by sparse bipartite graphs. The graphical representation of LDPC codes is so popular that most people refer to an LDPC code in terms of the structure of its factor graph. A *factor graph* is always a bipartite graph whose nodes are partitioned into the set of *variable nodes* and the set of *check nodes*. Symbols used for representing variable and check nodes are depicted in **Fig. 3.2**.

Figure 3.2: The circle represents a variable node and the square represents a check node.

The graph gives rise to a linear code of block length n and dimension *at least* $n-m$ in the following way: The n coordinates of the codewords are associated with the n variable nodes. The codewords are those vectors $(x_1, \ldots, x_n)$ such that all the parity-check equations are satisfied. **Fig. 3.3** gives an example.

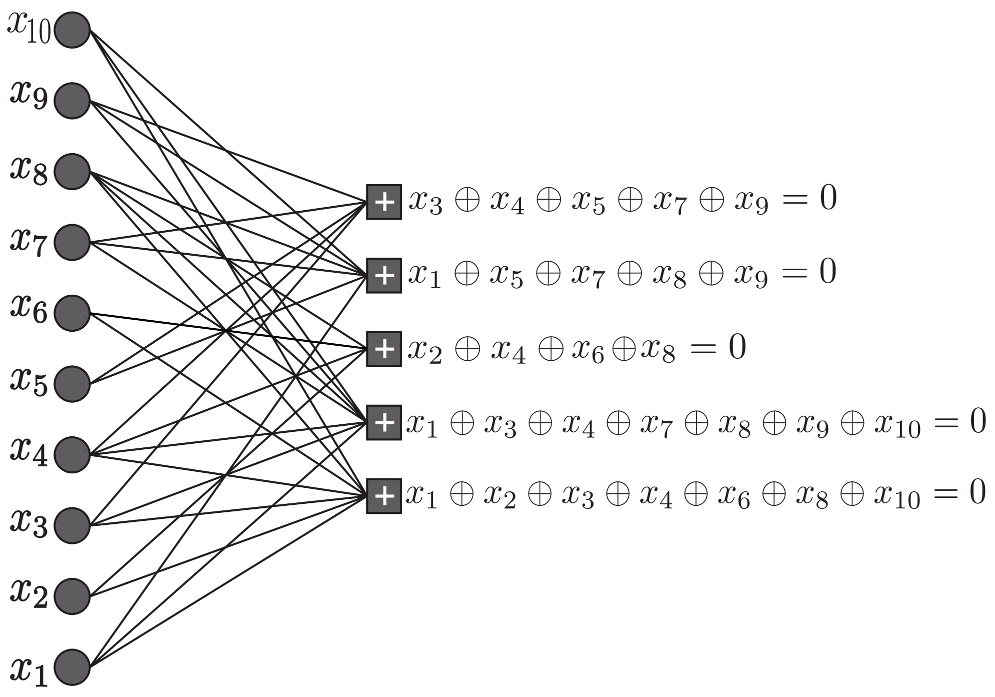

x_{10}
x_9
x_8
x_7
x_6
x_5
x_4
x_3
x_2
x_1
$x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_9 = 0$
$x_1 \oplus x_5 \oplus x_7 \oplus x_8 \oplus x_9 = 0$
$x_2 \oplus x_4 \oplus x_6 \oplus x_8 = 0$
$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_8 \oplus x_9 \oplus x_{10} = 0$
$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_{10} = 0$

Figure 3.3: Bipartite graph representing the irregular LDPC code of the parity-check matrix in **3.3**.

The graph representation is analogous to a matrix representation by looking at the adjacency matrix of the graph: let $\mathcal{H}$ be a binary $m \times n$-matrix in which the entry (i, j) is 1 if, and only if, the i-th check node is connected to the j-th variable node in the graph. Hence, each row indicates which bits participate in the corresponding parity-check equation, and each column indicates which parity-check equations the corresponding bit participates in.

Definition 3.1. The *degree* of a node is defined as the number of edges connected to that node. $\diamond$

Definition 3.2. The minimum cycle length of a graph is called the *girth* of the graph. $\diamond$

It is known that a *cycle* is a path on the graph such that the start vertex and the end vertex are the same. Since double edges are not allowed in Tanner graphs, i.e. there are not cycles of length 2, the *minimum girth* of a Tanner graph will be 4. That is, in bipartite graphs there are not odd-lengthed cycles.

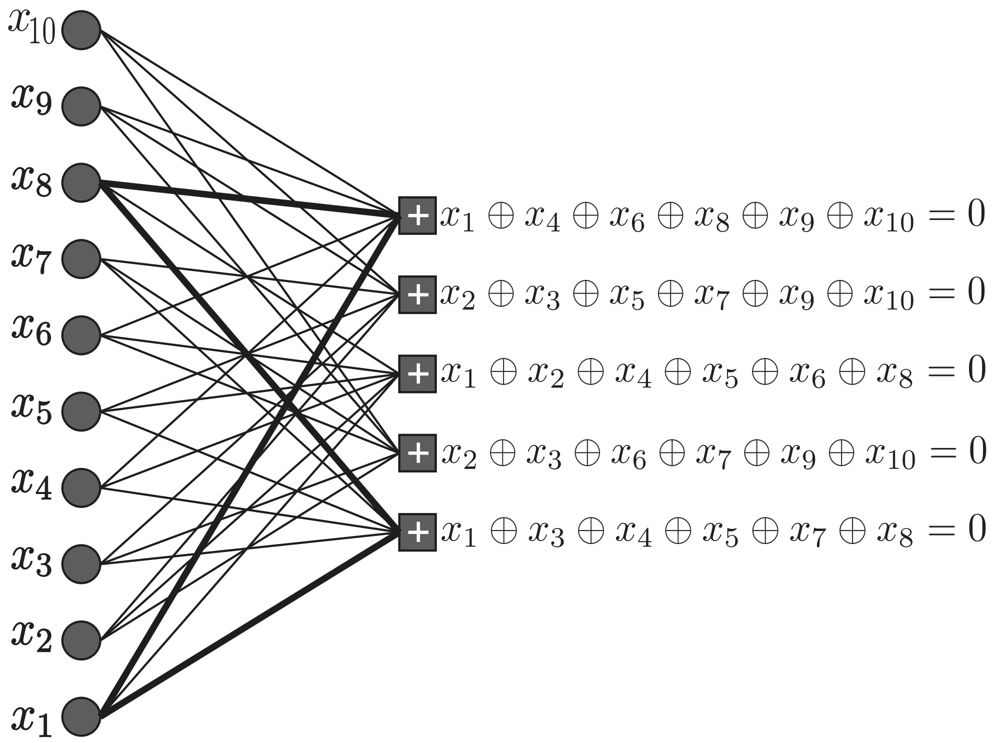

x_{10}
x_9
x_8
x_7
x_6
x_5
x_4
x_3
x_2
x_1
$x_1 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_9 \oplus x_{10} = 0$
$x_2 \oplus x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{10} = 0$
$x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8 = 0$
$x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_9 \oplus x_{10} = 0$
$x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8 = 0$

Figure 3.4: Bipartite graph representing the $(3, 6)$-regular LDPC code of example 3.1. The darkened edges represent a cycle of length 4.

Let E denote the number of edges in a bipartite graph. Also observe from the figures of this subsection that all edges are connected between the two sets. That is, the number of edges connected to variable nodes must be the same with the number of edges connected to check nodes. Recall that d_{v} denotes the variable node degree of a regular code and d_c denotes the corresponding check node degree. Hence, it follows that

$$E = m \cdot d_c = n \cdot d_{\mathrm{v}}. \qquad \textbf{3.4}$$

An important property is that the number of edges in the Tanner graph of a (d_{v},d_c)-regular LDPC code is $d_{\mathrm{v}} \cdot n$, where n is the length of the code. As n increases, the number of edges in the Tanner graph grows *linearly* in n.

Using the aforementioned equation in **3.2** we prove the second equality of **3.2**. Namely, using the fact that $\frac{m}{n} = \frac{d_{\mathrm{v}}}{d_c}$ we take $r = \frac{k}{n} \geq \frac{n-m}{n} = 1 - \frac{m}{n} = 1 - \frac{d_{\mathrm{v}}}{d_c}$.
We conclude this subsection stating that any linear code has a representation as a code associated to a bipartite graph. However, not every binary linear code has a representation by a *sparse* bipartite graph.[2] If it does, then the code is called an LDPC code. The sparsity of the graph structure is the key property that allows the algorithmic efficiency of LDPC codes. In Chapter 4, we will see that the graph representation of LDPC codes plays a vital role in decoding, since the most widely used decoding algorithm is in fact an algorithm for marginalising functions, which can be implemented using a certain graph representation. Additionally, it is essential for the graph to have as large a *girth* as possible so that the decoding algorithm works well.

[2]To be more precise, sparsity only applies to matrices. A matrix is called sparse if the number of non-zero elements about the 10% of the total entries.

3.3 Degree Distributions & Code Ensemble

Constructing provably good codes is difficult. A standard approach to show the existence of good codes is the probabilistic method: an *ensemble* of codes is "constructed" using some random procedure and one proves that good codes occur with positive probability within this ensemble. Often the probability is close to 1, that is almost all codes are good. This approach, used already by Shannon in his 1948 landmark paper, simplifies the code "construction" task enormously. An *ensemble* consists of all possible codes of length n and cardinality 2^k. We endow this set with a uniform probability distribution. How to sample from the *ensemble* will be discussed later.
Now consider the *ensemble* of regular LDPC codes with variable degree d_v, check degree d_c, and length n. If n is large enough, the average behaviour of almost all instances of this ensemble *concetrates* around the expected behaviour [6]. Hence, regular codes referred to by their variable and check degree and their length. When the performance and properties of sufficiently long regular LDPC codes are of interest, they are presented only by their variable and check node degree. For instance a (3,6)-LDPC code refers to a code with variable nodes of degree 3 and check nodes of degree 6. The *design rate* of this code is $\frac{1}{2}$. Before defining the *irregular ensemble* we will introduce the so called *degree distributions*.

Node Perspective

Assume that an LDPC code has length n and that the number of variable nodes of degree i is Λ_i, so that $\sum_i \Lambda_i = n$. Likewise, denote the number of check nodes of degree i by P_i so that $\sum_i P_i = m = n \cdot (1 - r)$ where r is the *design rate*, as stated previously.
Recall that the edge counts must match up, hence

$$\underbrace{\sum_i i\Lambda_i}_{\Lambda'(1)} = \underbrace{\sum_i iP_i}_{P'(1)} . \qquad \textbf{3.5}$$

It is convenient to introduce the following compact notation:

$$\Lambda(x) = \sum_{i\geq 2}^{\ell_{max}} \Lambda_i x^i, \quad P(x) = \sum_{i\geq 2}^{r_{max}} P_i x^i \qquad \textbf{3.6}$$

i.e., $\Lambda(x)$ and $P(x)$ are polynomials with non-negative expansions around zero whose integral coefficients are equal to the number of nodes of various degrees. From these definitions, we see immediately the following relationships:

$$\Lambda(1) = n, \qquad P(1) = m, \qquad r(\Lambda, P) = 1 - \frac{P(1)}{\Lambda(1)}. \tag{3.7}$$

We call Λ and P variable and check *degree distributions from a node perspective.* Often it is useful to use the *normalised* degree distributions:

$$L(x) = \frac{\Lambda(x)}{\Lambda(1)} = \sum_{i\geq 2}^{\ell_{max}} L_i x^i, \qquad R(x) = \frac{P(x)}{P(1)} = \sum_{i\geq 2}^{r_{max}} R_i x^i. \tag{3.8}$$

Since each of $\{L_i\}_{i\geq 2}^{\ell_{max}}$ and $\{R_i\}_{i\geq 2}^{r_{max}}$ is a probability mass function we must have that $\sum_{i\geq 2}^{\ell_{max}} L_i = 1$ and $\sum_{i\geq 2}^{r_{max}} R_i = 1$. Thus, a variable node will be of degree i, that is, it will be connected with i check nodes, with probability L_i. Likewise, a check node will be of degree j, that is, it will be connected with j variable nodes, with probability R_j.

Edge perspective

For the asymptotic analysis it is more useful to take on an edge perspective. Define

$$\lambda(x) = \sum_{i\geq 2}^{\ell_{max}} \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{L'(x)}{L'(1)}, \qquad \rho(x) = \sum_{i\geq 2}^{r_{max}} \rho_i x^{i-1} = \frac{P'(x)}{P'(1)} = \frac{R'(x)}{R'(1)}. \tag{3.9}$$

Note that $\lambda(x)$ and $\rho(x)$ are polynomials with non-negative expansions around zero. Some thought shows that λ_i/ρ_i is equal to the fraction of edges that connect to variable/check nodes of degree i. To clarify the aforementioned claim we present a relationship for λ_i:

$$\lambda(x) = \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{\sum_i \left(\frac{d}{dx}\Lambda_i x^i\right)}{\Lambda'(1)} = \frac{\sum_i i\Lambda_i x^{i-1}}{\Lambda'(1)} \Rightarrow \lambda_i = \frac{i\Lambda_i}{\Lambda'(1)}. \tag{3.10}$$

Likewise for ρ_i we obtain

$$\rho_i = \frac{iP_i}{P'(1)}. \tag{3.11}$$

In other words, λ_i/ρ_i is the probability that an edge chosen uniformly at random from the graph is connected to a variable/check node of degree i. We call $\lambda(x)$ and $\rho(x)$ the variable and check *degree distributions from an edge perspective.*

Furthermore, we also define the average variable and check degrees, call them ℓ_{avg} and r_{avg}, as:

$$\ell_{avg} = L^{'}(1) = \frac{1}{\int_0^1 \lambda(x)dx}, \qquad r_{avg} = R^{'}(1) = \frac{1}{\int_0^1 \rho(x)dx} \tag{3.12}$$

respectively, and the *design rate* is given by

$$r(\lambda, \rho) = 1 - \frac{\ell_{avg}}{r_{avg}} = 1 - \frac{L^{'}(1)}{R^{'}(1)} = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}. \tag{3.13}$$

The *design rate* is the rate of the code assuming that all constraints are *linearly independent.* Notice that the graph is characterised in terms of the fraction of edges of each degree and not the nodes of each degree.

In the beginning of this section we discussed about the *regular* ensemble. Similar to regular codes, it is shown [6] that the average behaviour of almost all instances of an ensemble of irregular codes is *concentrated* around its expected behaviour, when the code is large enough. Additionally, the expected behaviour converges to the cycle-free case [6].

Given the degree distribution of an LDPC code and its number of edges $E = \Lambda'(1) = P'(1)$, the number of variable nodes n is

$$n = E \sum_i \frac{\lambda_i}{i} = E \int_0^1 \lambda(x)dx, \tag{3.14}$$

and the number of check nodes m is

$$m = E \sum_i \frac{\rho_i}{i} = E \int_0^1 \rho(x)dx. \tag{3.15}$$

The previous two equations can be proved easily using **3.10** and **3.11**, respectively. Using the last relations for n and m, we can easily calculate the *design rate* as

$$r = 1 - \frac{\sum_i \frac{\rho_i}{i}}{\sum_i \frac{\lambda_i}{i}}. \tag{3.16}$$

Example 3.3. (Conversion from node to edge perspective). Consider the pair (Λ, P):

$$\Lambda(x) = 613x^2 + 202x^3 + 57x^4 + 84x^7 + 44x^8, \quad P(x) = 500x^6$$

with $\Lambda(1) = 1000$, $P(1) = 500$ and $\Lambda'(1) = P'(1) = 3000$. That is, there are 1000 variable nodes, 500 check nodes, and 3000 edges in the graph. We know that the variable node degree distribution from an edge perspective is given by $\lambda(x) = \sum_i \lambda_i x^{i-1}$. Hence, using from **3.10** the fact that $\lambda_i = \frac{i \cdot \Lambda_i}{3000}$, we have:

$$\lambda(x) = \frac{2 \cdot 613}{3000}x + \frac{3 \cdot 202}{3000}x^2 + \frac{4 \cdot 57}{3000}x^3 + \frac{7 \cdot 84}{3000}x^6 + \frac{8 \cdot 44}{3000}x^7.$$

Using the corresponding relation for ρ_i in **3.11** we obtain

$$\rho(x) = \frac{6 \cdot 500}{3000}x^5 = x^5.$$

□

Example 3.4. (Conversion from edge to node perspective). Assume now that the $\lambda(x)$ and $\rho(x)$ distributions of the aforementioned example are known and we wish to find the corresponding $\Lambda(x)$ and $P(x)$. Solving **3.10** and **3.11**, w.r.t. Λ_i and P_i, respectively, we have $\Lambda_i = \frac{\lambda_i \Lambda'(1)}{i}$ and $P_i = \frac{\rho_i P'(1)}{i}$. Therefore,

$$\begin{aligned}\Lambda(x) &= \frac{\frac{1226}{3000}\cdot 3000}{2}x^2 + \frac{\frac{3\cdot 202}{3000}\cdot 3000}{3}x^3 + \frac{\frac{4\cdot 57}{3000}\cdot 3000}{4}x^4 + \frac{\frac{7\cdot 84}{3000}\cdot 3000}{7}x^7 + \frac{\frac{8\cdot 44}{3000}\cdot 3000}{8}x^8 \\ &= 613x^2 + 202x^3 + 57x^4 + 84x^7 + 44x^8\end{aligned}$$

and

$$P(x) = \frac{\frac{6\cdot 500}{3000}\cdot 3000}{6}x^6 = 500x^6.$$

□

Finally, polynomials $\lambda(x)$ and $\rho(x)$ are *very important* since they determine:

- the code's *design rate*, $r = 1 - \frac{\sum_i \frac{\rho_i}{i}}{\sum_i \frac{\lambda_i}{i}}$.
- the code's *average performance.*

Additionally, from **3.14** it can be seen that, fixing the variable degree distribution of a code, the *number of edges* in the factor graph of such a code is *proportional* to n. This is the essential property of LDPC codes, which makes their decoding complexity *linear* with the code length given a fixed number of iterations. This is because the decoding is performed by passing messages along the edges of the graph, hence the complexity of one iteration is of the order of E. There are many different message-passing algorithms for LDPC codes. The purpose of next Chapter is to introduce and analyse the *sum-product* algorithm (or *belief propagation*).

Definition 3.3. (The standard ensemble LDPC(Λ,P) [1, p.78]). Given a degree distribution pair (Λ,P), define an *ensemble* of bipartite graphs LDPC(Λ,P) in the following manner. Each graph in LDPC(Λ,P) has $\Lambda(1)$ variable nodes and $P(1)$ check nodes. As state earlier, Λ_i variable nodes and P_i check nodes have degree i. A node of degree i has i sockets from which the i edges emanate, so that in total there are $\Lambda'(1) = P'(1)$ sockets on each side. Label the sockets on each side with the set $\left[\Lambda'(1)\right] = \{1, \cdots, \Lambda'(1)\}$ in some arbitrary but fixed way. Let σ be a a *permutation* on $\left[\Lambda'(1)\right]$. Associate to σ a bipartite graph by connecting the i-th socket on the variable side to the $\sigma(i)$-th socket on the check side. Letting σ run over the set of permutations on $\left[\Lambda'(1)\right]$ generates a set of bipartite graphs. Finally, we define a probability distribution over the set of graphs by placing the uniform probability distribution on the set of permutations. This is the *ensemble of bipartite graphs* LDPC(Λ, P). In the random graph literature this is what is called *configuration model.*

It remains to associate a code with every element of LDPC(Λ, P). We will do so by associating a parity-check matrix to each graph. Because of possible multiple edges and since the encoding is done over the field $\mathbb{F}_2$, we define the parity-check matrix $\mathcal{H}$ as the binary matrix that has a non-zero entry at row i and column j if the i-th check node is connected to the j-th variable node an *odd* number of times.

Since to every graph we can associate a code, we use these two terms interchangeably and we refer, e.g., to codes as elements of LDPC(Λ,P).

This is a subtle point: graphs are *labeled* (they have labeled sockets) and have a uniform probability distribution; the induced codes are unlabelled and the probability distributions is not necessarily the uniform one. Therefore, if in the sequel we say that we pick a code uniformly at random we really mean that we pick a graph at random from the ensemble of graphs and consider the induced code. $\diamond$

It can be shown that ensembles with a positive fraction of degree 1 variable nodes have non-zero bit error probability for *all* non-zero channel parameters even in the limit of infinite blocklengths: by our aforementioned definition of the ensemble there is a positive probability that two degree 1 variable nodes connect to the same check node and such a code contains codewords of weight 2. Therefore, we only consider ensembles *without degree 1* nodes.

How to sample from an LDPC ensemble

Let us assume that we have computed the optimal $(\lambda(x), \rho(x))$ and we wish to sample from this ensemble.

At first, we observe that $(\lambda(x), \rho(x))$ are the degree distributions from an *edge perspective*. A way to sample is to generate the columns of the parity-check matrix $\mathcal{H}$ column-by-column, generating a certain number of 1s in each column. In order to do this, we must compute the degree distributions from a node perspective.

Lemma 3.1. If $\lambda(x) = \sum_{i=2}^{\ell_{max}} \lambda_i x^{i-1}$, then $L(x) = \sum_{i=2}^{\ell_{max}} L_i x^i$ with

$$L_i = \frac{1}{\sum_j \frac{\lambda_j}{j}} \frac{\lambda_i}{i}, \quad \text{for } i = 2, \dots, \ell_{max}.$$

Proof. Recall that $\lambda(x) = \frac{L'(x)}{L'(1)}$. Thus,

$$\begin{aligned}
L(x) &= L'(1) \int_0^x \lambda(z) dz \\
&= L'(1) \int_0^x \sum_{i=2}^{\ell_{max}} \lambda_i z^{i-1} dz \\
&= L'(1) \sum_{i=2}^{\ell_{max}} \lambda_i \int_0^x z^{i-1} dz \\
&= L'(1) \sum_{i=2}^{\ell_{max}} \lambda_i \cdot \frac{x^i}{i} \\
&= \sum_{i=2}^{\ell_{max}} \left(L'(1) \cdot \frac{\lambda_i}{i} \right) x^i \\
&= \sum_{i=2}^{\ell_{max}} L_i x^i.
\end{aligned}$$

Since $\{L_i\}_{i=2}^{\ell_{max}}$ is a probability mass function, we must have that $\sum_{i=2}^{\ell_{max}} L_i = 1$ yielding that $L'(1) = \frac{1}{\sum_j \frac{\lambda_j}{j}}$. Hence, a variable node will be of degree i, that is, it will be connected with i check nodes, with probability L_i. ∎

Analogous expression can be derived and proved for check node distribution. That is, $P_i = \frac{1}{\sum_j \frac{\rho_j}{j}} \frac{\rho_i}{i}$ for $i = 2, \dots, r_{max}$.

In order to generate a random parity-check matrix, given $(\lambda(x), \rho(x))$, and the length of the codeword, n, *we first compute* $L(x)$ and then proceed as in definition 3.3. Thus,

1. we compute the total number of edges, E_{total},

2. we randomly permute the set $\left[\Lambda'(1)\right]$,

3. for *edge* $= 1 : E_{total}$, we "connect" the edge socket of the variable node side with the permuted(edge) socket of the check node side. *The most important point here is to associate the variable node and check node sockets with the corresponding variable and check nodes.* After the association, it is easy to put "1" at the corresponding position. That is, if i-th check node is connected to the j-th variable node an *odd* number of times we put "1" in (i, j) position of the parity check matrix. Otherwise, we put 0.

Chapter 4

Decoding of LDPC codes

4.1 Belief Propagation Decoding

Maximum-likelihood (ML) decoding of LDPC codes has usually exponential complexity. We apply the belief propagation (BP) algorithm, which can be viewed as applying Bayes' rule locally and iteratively to calculate approximate marginal *a posteriori* probabilities. BP is practical since it has linear decoding complexity per iteration. In the cases where the Tanner graph is cycle free, it can be proved that BP algorithm computes *exactly* the marginal *a posteriori* probabilities. Otherwise, BP provides very good approximations.

4.1.1 Factor Graphs

In this section, we follow the derivation of Chapter 2 from Richardson & Urbanke's book [1]. Factor graphs can be used for the representation of the factorization of a multi-variable function into a product of sub-functions. A factor graph is bipartite, i.e, the set of vertices is partitioned into two groups, the set of nodes corresponding to variables (depicted with circles) and the set of nodes corresponding to factors (depicted with squares). In other words, there exists no edge that connects two factor nodes or two variable nodes. In the following derivation, the factor graph is a (bipartite) tree, i.e., there are no cycles in the graph. Marginals can be computed efficiently by message-passing algorithms, if the factor graphs are trees. To simplify the calculation of a factorised function's marginal, we will take advantage of the distributive law. Consider that we want to compute the expression $\sum_{i,j} x_i y_j = (\sum_i x_i)(\sum_j y_j)$ and the cardinality of i and j is n, it is easy to observe that, using the distributive law, we have a computational reduction from n^2 to $2n$.

Let $\mathcal{I} \triangleq (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ and a function f that can be factorised as follows:

$$f(\mathcal{I}) = f_1(x_1, x_2, x_3) f_2(x_1, x_4) f_3(x_1, x_6, x_7) f_4(x_4, x_5) f_5(x_4) f_6(x_2) \qquad \textbf{4.1}$$

and suppose that we wish to compute the marginal of f with respect to variable x_1, then

$$\begin{aligned} f_{X_1}(x_1) &= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5}\sum_{x_6}\sum_{x_7} f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \\ &= \sum_{\sim x_1} f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \\ &= \sum_{\sim x_1} f_1(x_1, x_2, x_3) f_2(x_1, x_4) f_3(x_1, x_6, x_7) f_4(x_4, x_5) f_5(x_4) f_6(x_2) \qquad \textbf{4.2} \end{aligned}$$

where $\sim x_1$ denotes the sum over all variables except x_1. Assuming that all variables take values in a finite discrete alphabet $\mathcal{X}$, brute force method requires $\Theta(|\mathcal{X}|^7)$ operations for the computation $f_{X_1}(x_1)$.
Applying the distributive law in **4.2**, we have:

$$\begin{aligned} f_{X_1}(x_1) &= \sum_{\sim x_1} f_1(x_1, x_2, x_3) f_2(x_1, x_4) f_3(x_1, x_6, x_7) f_4(x_4, x_5) f_5(x_4) f_6(x_2) \\ &= \left[\sum_{x_2}\left(f_6(x_2)\sum_{x_3} f_1(x_1, x_2, x_3)\right)\right]\left[\sum_{x_4}\left(f_5(x_4) f_2(x_1, x_4)\sum_{x_5} f_4(x_4, x_5)\right)\right] \\ &\quad \left[\sum_{x_6, x_7} f_3(x_1, x_6, x_7)\right]. \qquad \textbf{4.3} \end{aligned}$$

In **4.3** the first factor can be efficiently evaluated in the following manner. For each value of variable x_2 and x_1 fixed, determine the sum over variable x_3 (requires $\Theta(|\mathcal{X}|)$ operations), multiply by $f_6(x_2)$ and sum over x_2. Hence, the first term will be computed in $\Theta(|\mathcal{X}|^2)$ operations. In the same fashion, the computation of the second factor requires $\Theta(|\mathcal{X}|^2)$ operations. The last factor is a sum over variables x_6 and x_7, consequently, this factor requires $\Theta(|\mathcal{X}|^2)$ operations. Finally, the overall task to compute the marginal $f_{X_1}(x_1)$ requires $\Theta(|\mathcal{X}|^3)$ operations, since variable x_1 has $|\mathcal{X}|$ possible values. We can observe that applying the distributive law leads to more efficient computations.

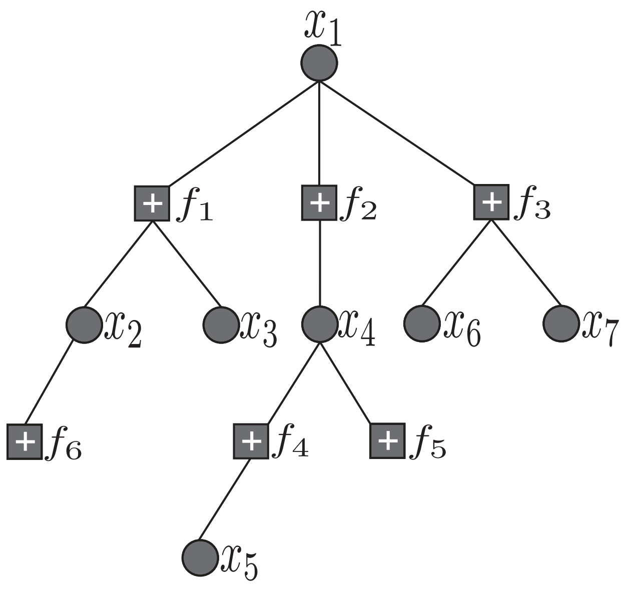


Figure 4.1: Factor Graph of the factorisation of function f with respect to variable x_1.

Likewise, for marginal of f w.r.t. variables x_2, x_3, x_4 etc. For instance,

$$\begin{aligned}
f_{X_2}(x_2) &= \sum_{\sim x_2} f_1(x_1,x_2,x_3)f_2(x_1,x_4)f_3(x_1,x_6,x_7)f_4(x_4,x_5)f_5(x_4)f_6(x_2)\\
&= \underbrace{f_6(x_2)\Bigg[\sum_{x_1}\Bigg(\underbrace{\sum_{x_4} f_2(x_1,x_4)f_5(x_4)\underbrace{\sum_{x_5} f_4(x_4,x_5)}_{|\mathcal{X}|}}_{|X^2|}\Bigg)\Bigg(\underbrace{\sum_{x_6,x_7} f_3(x_1,x_6,x_7)}_{|\mathcal{X}^2|}\Bigg)\Bigg(\underbrace{\sum_{x_3} f_1(x_1,x_2,x_3)}_{|\mathcal{X}|}\Bigg)\Bigg]}_{|\mathcal{X}^3|}.
\end{aligned}$$

$$\begin{aligned}
f_{X_4}(x_4) &= \sum_{\sim x_4} f_1(x_1,x_2,x_3)f_2(x_1,x_4)f_3(x_1,x_6,x_7)f_4(x_4,x_5)f_5(x_4)f_6(x_2)\\
&= \underbrace{f_5(x_4)\sum_{x_1}\Bigg(\underbrace{f_2(x_1,x_4)\Bigg(\sum_{x_2}\Big(f_6(x_2)\underbrace{\sum_{x_3} f_1(x_1,x_2,x_3)}_{|\mathcal{X}|}\Big)}_{|\mathcal{X}|^2}\Bigg)\Bigg(\underbrace{\sum_{x_5} f_4(x_4,x_5)}_{|\mathcal{X}|}\Bigg)\Bigg(\underbrace{\sum_{x_6,x_7} f_3(x_1,x_6,x_7)}_{|\mathcal{X}|^2}\Bigg)\Bigg)}_{|\mathcal{X}|^3}.
\end{aligned}$$

Hence, the overall task to compute the marginals $f_{X_2}(x_2)$ and $f_{X_4}(x_4)$ requires $\Theta(|\mathcal{X}|^4)$ operations, since variables x_2 and x_4 have $|\mathcal{X}|$ possible values.

Example 4.1. Let $\mathcal{C}$ be a binary linear block code defined by the generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

and parity check matrix

$$\begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \mathcal{H} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \end{array} \qquad \mathbf{4.4}$$

Recall that matrix **G** generates a linear code since it has full row rank. We define $\mathbb{F}_2 = \{0,1\}$ and $\mathbf{x} = (x_1, \dots, x_5)$. Consider the function $f(x_1, \dots, x_5)$ from $\mathbb{F}_2^5$ to $\{0,1\} \subset \mathbb{R}$ that is defined by

$$f(x_1, \dots, x_5) = \mathbb{1}_{\{\mathbf{x} \in \mathbf{C}\}} = \begin{cases} 1, & if\ \mathbf{H}\mathbf{x}^T = \mathbf{0}, \\ 0, & otherwise. \end{cases} \qquad \mathbf{4.5}$$

Function f is called *code membership* function, since it tests whether a particular word **x** is a member of the code or not. Hence, due to **4.5**, function f can be factorised as

$$f(x_1, \dots, x_5) = \mathbb{1}_{\{x_3 + x_4 = 0\}} \cdot \mathbb{1}_{\{x_1 + x_2 + x_3 + x_5 = 0\}}. \qquad \mathbf{4.6}$$

The Factor Graph of f is also called the Tanner graph of $\mathcal{H}$ and is depicted in **Fig. 4.2**. Also, notice that the following Tanner graph is cycle free. That is, for each x_i, there is not a path that starts from x_i that leads to itself. □

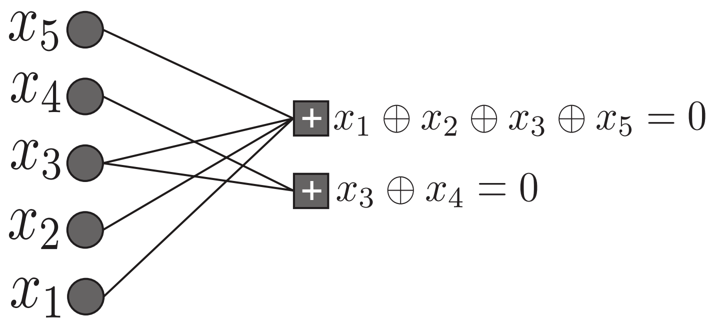


Figure 4.2: Factor graph representation of the code membership function based on the factorisation in 4.6.

4.1.2 Message Passing

Consider a factorization of a generic function g. Since the factor graph of g is a bipartite tree, g can be factorised as

$$g(z,\ldots) = \prod_{k=1}^{K}\Big[g_k(z,\ldots)\Big],\ K \in \mathbb{Z}^*, \tag{4.7}$$

with the following crucial property: z appears in each of the factors g_k, but all other variables appear in only one factor (consequence of the event that factor graph of g is a tree). Hence, due to 4.7, the factorisation in 4.1 can be written as

$$f(x_1,\ldots,x_7) = \underbrace{\Big[f_1(x_1,x_2,x_3)f_6(x_2)\Big]}_{g_1(x_1,\ldots)}\underbrace{\Big[f_2(x_1,x_4)f_4(x_4,x_5)f_5(x_4)\Big]}_{g_2(x_1,\ldots)}\underbrace{\Big[f_3(x_1,x_6,x_7)\Big]}_{g_3(x_1,\ldots)}. \tag{4.8}$$

We observe that the aforementioned property holds. That is, x_1 appears in each of the three factors, but all other variables, appear in only one factor.
Applying the distributive law in 4.7, we obtain

$$\sum_{\sim z} g(z,\ldots) = \underbrace{\sum_{\sim z}\prod_{k=1}^{K}\Big[g_k(z,\ldots)\Big]}_{\text{marginal of product}} = \underbrace{\prod_{k=1}^{K}\sum_{\sim z}\Big[g_k(z,\ldots)\Big]}_{\text{product of marginals}}. \tag{4.9}$$

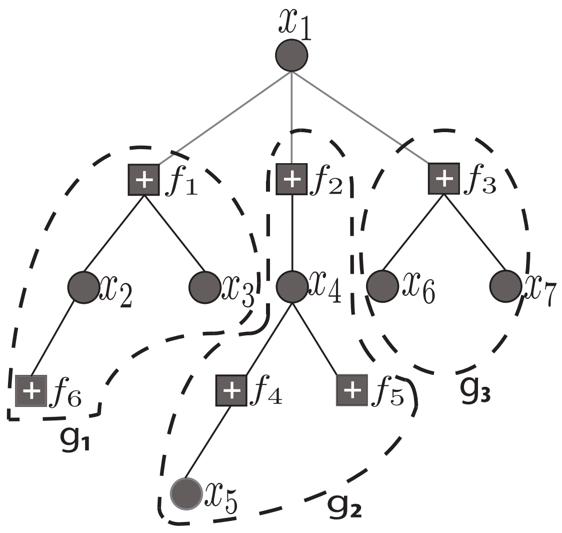


Figure 4.3: Factor graph representation of expansion in 4.8.

That is, 4.3 can be equivalently written as

$$\begin{aligned} f_{X_1}(x_1) &= \sum_{\sim x_1} f_1(x_1,x_2,x_3)f_2(x_1,x_4)f_3(x_1,x_6,x_7)f_4(x_4,x_5)f_5(x_4)f_6(x_2) \\ &= \sum_{\sim x_1}\Bigg(\underbrace{\Big[f_1(x_1,x_2,x_3)f_6(x_2)\Big]}_{g_1(x_1,\ldots)}\underbrace{\Big[f_2(x_1,x_4)f_4(x_4,x_5)f_5(x_4)\Big]}_{g_2(x_1,\ldots)}\underbrace{\Big[f_3(x_1,x_6,x_7)\Big]}_{g_3(x_1,\ldots)}\Bigg) \\ &= \Bigg[\sum_{\sim x_1} f_6(x_2)f_1(x_1,x_2,x_3)\Bigg]\Bigg[\sum_{\sim x_1} f_5(x_4)f_2(x_1,x_4)f_4(x_4,x_5)\Bigg] \\ &\quad \Bigg[\sum_{\sim x_1} f_3(x_1,x_6,x_7)\Bigg]. \end{aligned} \tag{4.10}$$

We can go further applying the same recursively to each of the terms $g_k(z,\ldots)$, until no further expansion can be applied. Each g_k is itself a product of factors and since the factor graph is a bipartite tree, g_k must in turn have a generic factorization of the form

$$g_k(z,\ldots) = \underbrace{h(z,z_1,\ldots,z_K)}_{\text{kernel}}\prod_{k=1}^{K}\Big[h_k(z_k,\ldots)\Big]. \tag{4.11}$$

From 4.8, we observe that the first and the second factor (i.e. $g_1(x_1,\ldots)$ and $g_2(x_1,\ldots)$)

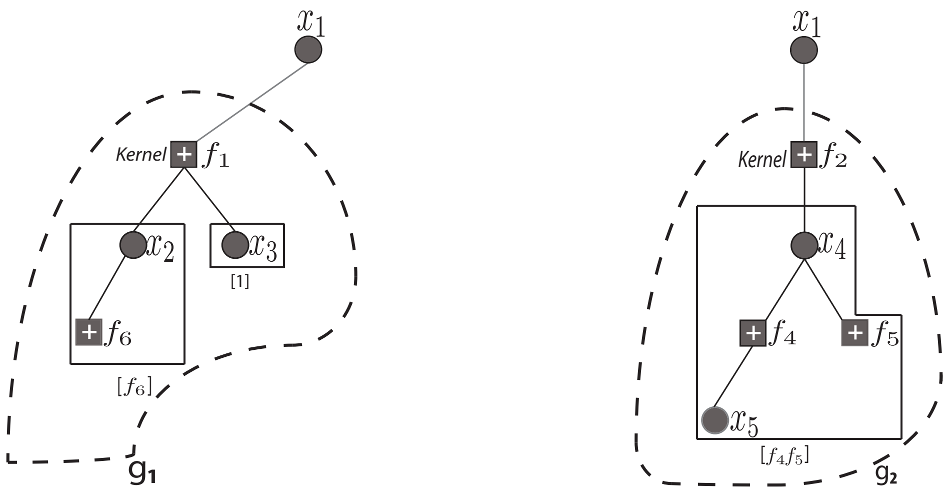
can be further expanded but not the third one. According to **4.11**, we have

$$g_1(x_1,\ldots) = \Big[f_1(x_1,x_2,x_3)f_6(x_2)\Big] = \underbrace{f_1(x_1,x_2,x_3)}_{\text{kernel}}\cdot\underbrace{\Big[f_6(x_2)\Big]}_{x_2}\cdot\underbrace{\Big[1\Big]}_{x_3}, \qquad \textbf{4.12}$$

$$g_2(x_1,\ldots) = \Big[f_2(x_1,x_4)f_4(x_4,x_5)f_5(x_4)\Big] = \underbrace{f_2(x_1,x_4)}_{\text{kernel}}\cdot\underbrace{\Big[f_4(x_4,x_5)f_5(x_4)\Big]}_{x_4}, \qquad \textbf{4.13}$$

and

$$g_3(x_1,\ldots) = f_3(x_1,x_6,x_7) = \underbrace{f_3(x_1,x_6,x_7)}_{\text{kernel}}\cdot\underbrace{[1]}_{x_6}\cdot\underbrace{[1]}_{x_7}. \qquad \textbf{4.14}$$



(a) (b)

Figure 4.4: Factor graph representation of the expansion of factors $g_1(x_1,\cdots)$ and $g_2(x_1,\cdots)$ respectively.

Consequently,

$$\begin{aligned} g_k(z) &= \sum_{\sim z} g_k(z,\ldots) = \sum_{\sim z} h(z,z_1,\ldots,z_K)\prod_{k=1}^{K}\Big[h_k(z_k,\ldots)\Big] \\ &= \sum_{\sim z} h(z,z_1,\ldots,z_K)\underbrace{\prod_{k=1}^{K}\Big[\sum_{\sim z_k} h_k(z_k,\ldots)\Big]}_{\text{product of marginals}}. \qquad \textbf{4.15} \end{aligned}$$

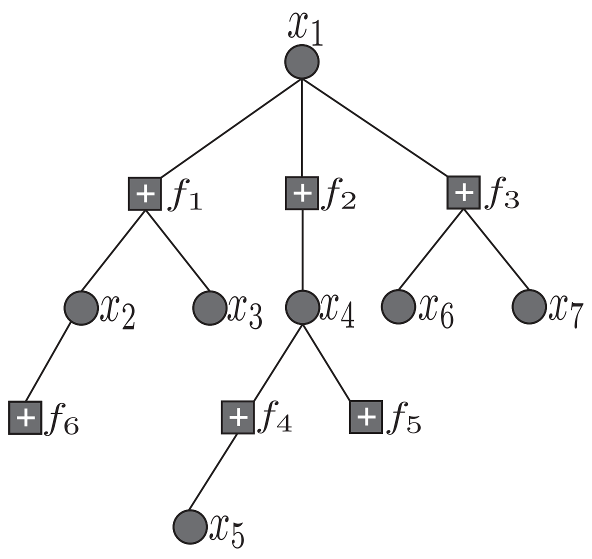
The desired marginal $\sum_{\sim z} g_k(z, \ldots)$ can be computed efficiently by multiplying the kernel function $h(z, z_1, \ldots, z_K)$ with the individual marginals and summing over all variables except z. In general, nodes in the graph compute marginals (messages), which are functions over $\mathcal{X}$. This recursive splitting continues until we have reached the leaves of the tree. Whereas, the marginal calculation follows the recursive splitting *in reverse*.

4.1.3 Marginalisation *via* Message Passing

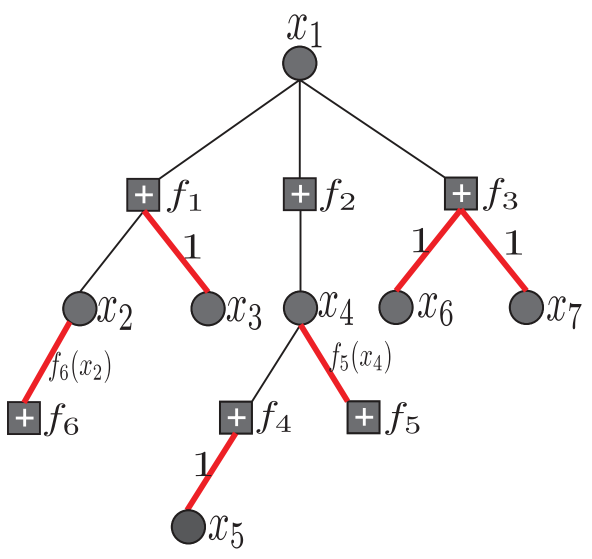
In the previous subsection, we discussed how the marginalisation problem can be recursively broken into smaller tasks due to the tree structure of the factor graph. Therefore, a function (or check) leaf node has the generic form $g_k(z)$, so that $g_k(z) = \sum_{\sim z} g_k(z)$. This means that the initial message sent by a function leaf node is the function itself. That is, in **4.13** we can further decompose the factor $[f_4(x_4, x_5) f_5(x_4)]$ according to **4.11** and we'll have $f_5(x_4) = \sum_{\sim x_4} f_5(x_4)$. The initial message for a variable leaf node is 1 as we can easily observe from **4.12**. In the sequel, we will state the message passing algorithm which will help us compute efficiently marginals.

Message Passing Algorithm

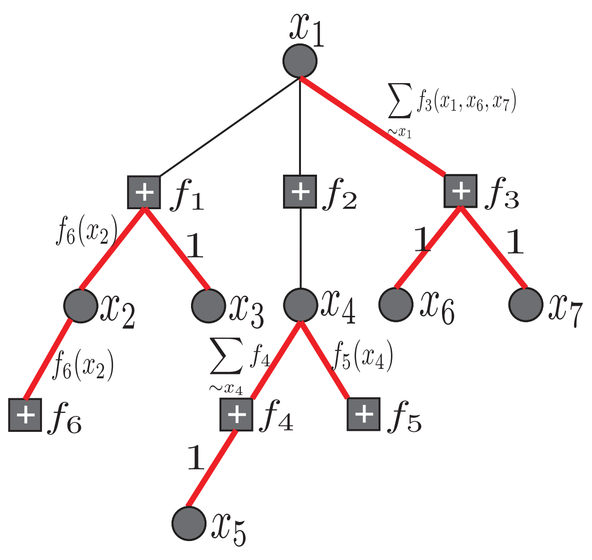
Messages signify marginals of parts of the function and are combined in order to calculate the marginal of the global function. Message passing originates at leaf nodes. The message passing algorithm works in a straightforward way. Messages are passed up the tree, as soon as a parent node has received all the incoming (child) messages. The message-passing algorithm ends when the last parent node is the root node of the tree where the final message multiplications happens.
Let us consider the factorisation of function f as shown in **4.1** and implement the marginalisation according to rule in **4.15** for function (check) nodes and point-wise multiplication for variable nodes. The initialisation step for both the factor and the variable nodes is depicted in **Fig. 4.5(b)**. The initial messages sent along the edges are: $\mu_{x_6 \to f_3}(x_6) = 1$, $\mu_{x_7 \to f_3}(x_7) = 1$, $\mu_{x_5 \to f_4}(x_5) = 1$, $\mu_{f_6 \to x_2}(x_2) = \sum_{\sim x_2} f_6(x_2) = f_6(x_2)$ and $\mu_{x_3 \to f_1}(x_3) = 1$. In the next time step, factor node f_3 and f_4 as well as variable node x_2, have received all the incoming (child) messages, hence, the corresponding outgoing messages from these nodes can be passed up the tree. Therefore, as depicted in **Fig. 4.5(c)**, $\mu_{x_2 \to f_1}(x_2) = f_6(x_2)$, $\mu_{f_4 \to x_4}(x_4) = \sum_{\sim x_4} f_4(x_4, x_5)$ and $\mu_{f_3 \to x_1}(x_1) = \sum_{\sim x_1} f_3(x_1, x_6, x_7)$.



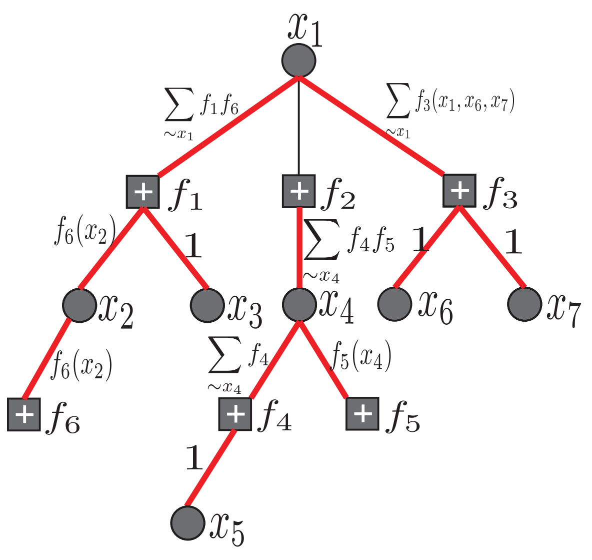
(a)



(b)



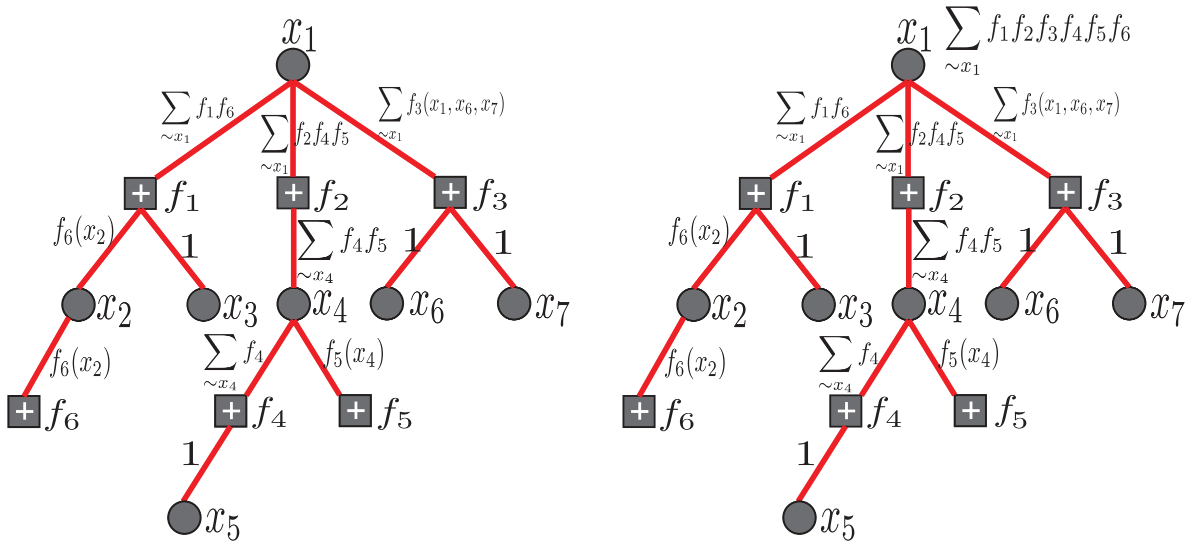
(c)



(d)

In the sequel (see **Fig. 4.5(d)**), variable node x_4 and factor node f_1 have received all incoming child messages, therefore, the corresponding outgoing messages will be: $\mu_{x_4 \to f_2}(x_4) = f_5(x_4) \sum_{\sim x_4} f_4(x_4, x_5) = \sum_{\sim x_4} f_4(x_4, x_5) f_5(x_4)$ and $\mu_{f_1 \to x_1}(x_1) = \sum_{\sim x_1} f_1(x_1, x_2, x_3) f_6(x_2)$. Afterwards, factor node f_2 has received all of its incoming messages, hence, as shown in **Fig. 4.5(e)**, $\mu_{f_2 \to x_1}(x_1) = \sum_{\sim x_1} f_2(x_1, x_4) \sum_{\sim x_4} f_4(x_4, x_5) f_5(x_4) = \sum_{\sim x_1} f_2(x_1, x_4) f_4(x_4, x_5) f_5(x_4)$.

Finally, root node x_1, has received all incoming child messages, therefore[1]
$f_{X1}(x_1) = \sum_{\sim x_1} f_1 f_2 f_3 f_4 f_5 f_6$.



x_1
$\sum_{\sim x_1} f_1 f_2 f_3 f_4 f_5 f_6$
$\sum_{\sim x_1} f_1 f_6$
$\sum_{\sim x_1} f_2 f_4 f_5$
$\sum_{\sim x_1} f_3(x_1, x_6, x_7)$
f_1
f_2
f_3
$f_6(x_2)$
1
$\sum_{\sim x_4} f_4 f_5$
x_2
x_3
x_4
x_6
x_7
$\sum_{\sim x_4} f_4$
$f_5(x_4)$
f_6
f_4
f_5
x_5

(e) (f)

Figure 4.5: Steps of Single Message Passing (SMP) Algorithm for computing marginal $f(x_1)$.

Bidirectional Message Passing

In bit-wise MAP decoding, we need to compute marginals over all variables. If we apply the previous single message passing algorithm we need to make the aforementioned derivation for each variable x_i, i.e., to draw for each variable, the corresponding tree rooted in this variable and execute the single message passing algorithm with the same way as in **Fig. 4.5**. It is easy to see, however, that this is computationally demanding.

[1]Notice that in **Fig. 4.5**, when we write $\sum_{\sim x_1} f_1 f_6$ we mean $\sum_{\sim x_1} f_1(x_1, x_2, x_3) f_6(x_2)$. We omit x_is so that the shapes be more clear.

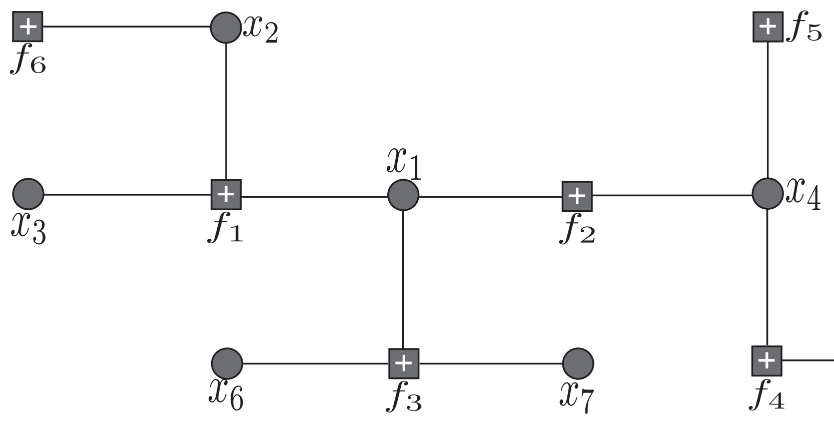
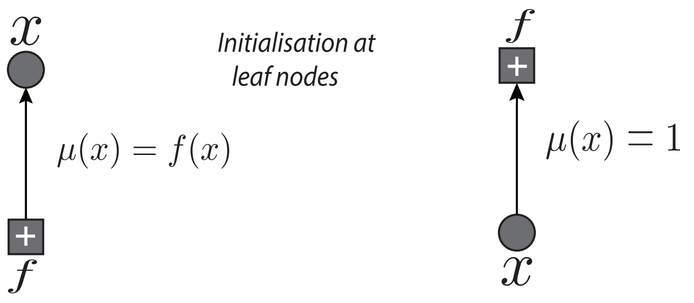
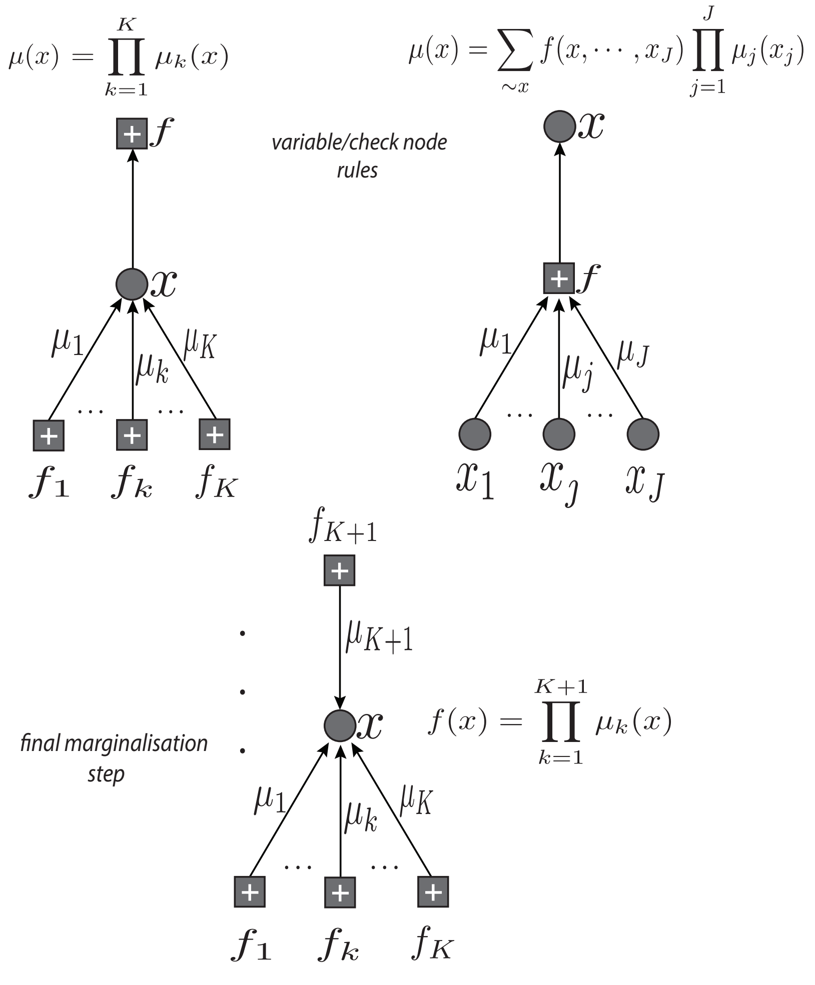


Figure 4.6: Equivalent Factor graph representation of **Fig. 4.5(a)** without hanging from a root node.

Moreover, SMP algorithm does not depend on which x_i is the root of the tree, i.e., the factor graph remains a tree. Hence, we can spread the factor graph as shown in **Fig. 4.6** and calculate all marginals simultaneously on a single tree. The marginalisation task starts from *all* leaf nodes. In our case, the leaves are: f_6, x_3, x_6, x_7, x_5 and f_5. Then, for every edge we compute the outgoing message along this edge as soon as we have received all the incoming messages along all other edges that connect to the given node. We continue in the same fashion, until a message has been sent in both directions along every edge. Hence, we have computed *all* marginals. Since the messages represent probabilities or beliefs, the algorithm is also known as the *belief propagation* (BP) algorithm. The complexity of BP algorithm is roughly the same as that of the SMP algorithm, but SMP computes a marginal w.r.t. x_i, whereas, BP computes all marginals.



(a)



$\mu(x) = \prod_{k=1}^{K} \mu_k(x)$
f
variable/check node rules
$\mu(x) = \sum_{\sim x} f(x, \cdots, x_J) \prod_{j=1}^{J} \mu_j(x_j)$
x
μ_1
μ_k
μ_K
f_1
f_k
f_K
μ_j
μ_J
x_1
x_j
x_J
f_{K+1}
μ_{K+1}
final marginalisation step
$f(x) = \prod_{k=1}^{K+1} \mu_k(x)$

(b)

Figure 4.7: Message-Passing Rules for BP algorithm.

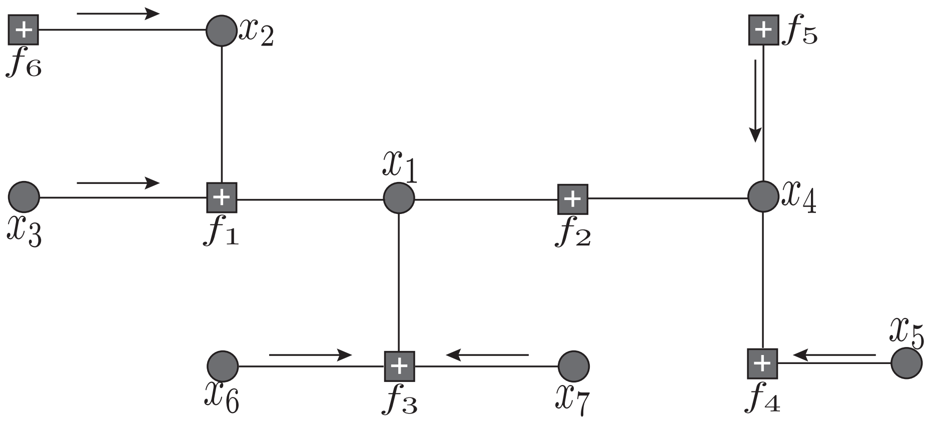
Example 4.2. Consider the factorisation in **4.1** and the corresponding factor graph as depicted in **Fig. 4.6**. In order to give the intuition of how BP algorithm works, we will present a step-by-step derivation. Recall that, the leaves are f_6, x_3, x_6, x_7, x_5 and f_5.

- **Round 1**

  According to rule in **Fig. 4.7(a)**, leaf nodes will sent the messages:

  1. $\mu_{f_6 \to x_2}(x_2) = f_6(x_2) = \sum_{\sim x_2} f_6(x_2)$.
  2. $\mu_{f_5 \to x_4}(x_4) = f_5(x_4) = \sum_{\sim x_4} f_5(x_4)$.
  3. $\mu_{x_3 \to f_1}(x_3) = 1$.

4. $\mu_{x_6 \to f_3}(x_6) = 1$.
5. $\mu_{x_7 \to f_3}(x_7) = 1$.
6. $\mu_{x_5 \to f_4}(x_5) = 1$.

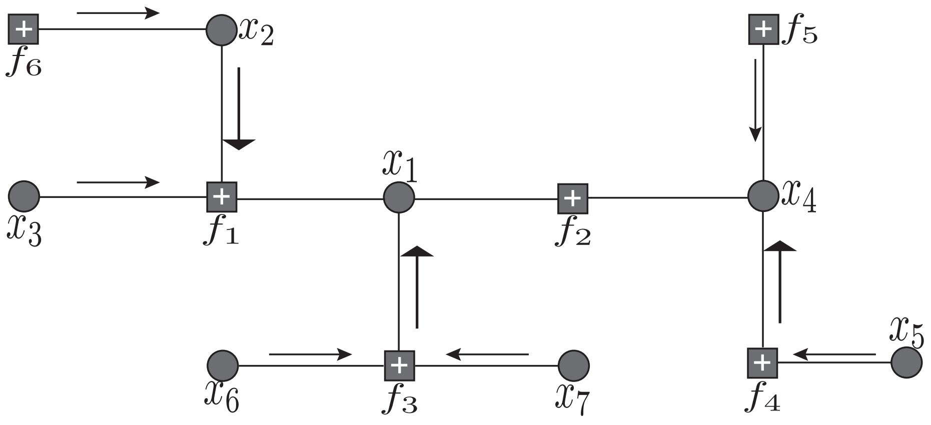


(a)

- **Round 2**

  According to the two rules on the top of **Fig. 4.7(b)** we have:

1. $\mu_{x_2 \to f_1}(x_2) = \mu_{f_6 \to x_2}(x_2) = f_6(x_2)$.
2. $\mu_{f_4 \to x_4}(x_4) = \sum_{\sim x_4} f_4(x_4, x_5) \cdot 1$.
3. Since f_3 has received all of its incoming messages, we have $\mu_{f_3 \to x_1}(x_1) = \sum_{\sim x_1} f_3(x_1, x_6, x_7) \cdot 1 \cdot 1$.

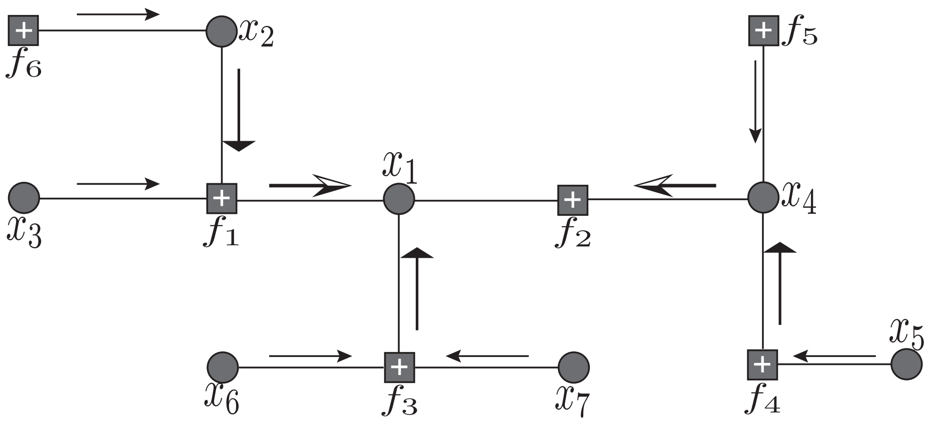


(b)

- **Round 3**

  Notice that *only* check node f_1 and variable node x_4 have received all of their incoming messages. Therefore,

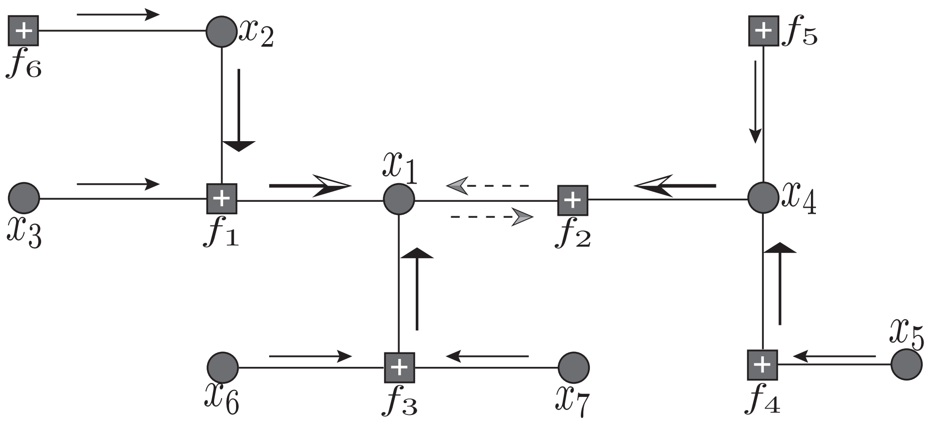
  1. $\mu_{f_1 \to x_1}(x_1) = \sum_{\sim x_1} f_1(x_1, x_2, x_3) \cdot f_6(x_2) \cdot 1.$
  2. $\mu_{x_4 \to f_2}(x_4) = \sum_{\sim x_4} f_4(x_4, x_5) \cdot f_5(x_4).$



(c)

- **Round 4**

  1. $\mu_{f_2 \to x_1}(x_1) = \sum_{\sim x_1} f_2(x_1, x_4) \cdot \mu_{x_4 \to f_2}(x_4) = \sum_{\sim x_1} f_2(x_1, x_4) f_4(x_4, x_5) f_5(x_4).$
  2. Since x_1 has received the incoming messages from f_1 and f_3, it forwards its outgoing message to check node f_2. Hence,
     $\mu_{x_1 \to f_2}(x_1) = \mu_{f_3 \to x_1}(x_1) \cdot \mu_{f_1 \to x_1}(x_1) = \sum_{\sim x_1} f_1(x_1, x_2, x_3) f_3(x_1, x_6, x_7) f_6(x_2).$

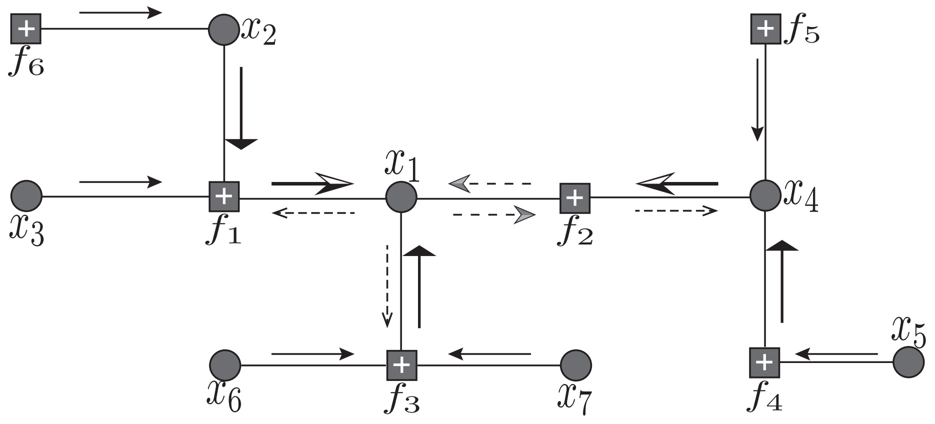


(d)

- **Round 5**

  Note that at this point, it can be proved [10, p.59] that we are able to compute the marginal $f_{X_1}(x_1)$ by simple point-wise multiplication of the messages over the edge (x_1,f_2). That is, $f_{X_1}(x_1) = \mu_{f_2\to x_1}(x_1) \cdot \mu_{x_1\to f_2}(x_1)$. Equivalently, we can omit the aforementioned procedure and apply at each variable node, the rule on the bottom of **Fig. 4.7(b)** as soon as we finish with the remaining rounds. At this round, the outgoing messages will be:

  1. $\mu_{x_1\to f_1}(x_1) = \mu_{f_2\to x_1}(x_1) \cdot \mu_{f_3\to x_1}(x_1) = \sum_{\sim x_1} f_2(x_1,x_4) f_3(x_1,x_6,x_7) f_4(x_4,x_5) f_5(x_4)$.
  2. $\mu_{x_1\to f_3}(x_1) = \mu_{f_1\to x_1}(x_1) \cdot \mu_{f_2\to x_1}(x_1) = \sum_{\sim x_1} f_1(x_1,x_2,x_3) f_2(x_1,x_4) f_4(x_4,x_5) f_5(x_4) f_6(x_2)$.
  3. $\mu_{f_2\to x_4}(x_4) = \sum_{\sim x_4} f_2(x_1,x_4) \mu_{x_1\to f_2}(x_1)$.

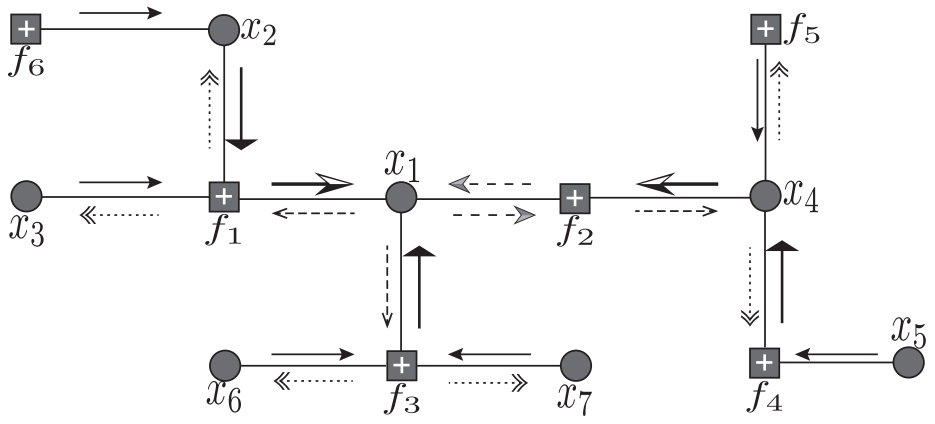


(e)

- **Round 6**

  There are six outgoing messages, since there are six “available” nodes. Hence,

  1. $\mu_{f_3\to x_6}(x_6) = \sum_{\sim x_6} f_3(x_1,x_6,x_7) \cdot \mu_{x_1\to f_3}(x_1) \cdot 1 = \sum_{\sim x_6} f_1 f_2 f_3 f_4 f_5 f_6$.
  2. $\mu_{f_3\to x_7}(x_7) = \sum_{\sim x_7} f_3(x_1,x_6,x_7) \cdot \mu_{x_1\to f_3}(x_1) \cdot 1 = \sum_{\sim x_7} f_1 f_2 f_3 f_4 f_5 f_6$.
  3. $\mu_{f_1\to x_3}(x_3) = \sum_{\sim x_3} f_1(x_1,x_2,x_3) \cdot \mu_{x_2\to f_1}(x_2) \cdot \mu_{x_1\to f_1}(x_1) = \sum_{\sim x_3} f_1 f_2 f_3 f_4 f_5 f_6$.
  4. $\mu_{f_1\to x_2}(x_2) = \mu_{x_3\to f_1}(x_3) \cdot \mu_{x_1\to f_1}(x_1) = \sum_{\sim x_2} f_1 f_2 f_3 f_4 f_5$.
  5. $\mu_{x_4\to f_5}(x_4) = \mu_{f_4\to x_4}(x_4) \cdot \mu_{f_2\to x_4}(x_4)$.
  6. $\mu_{x_4\to f_4}(x_4) = \mu_{f_2\to x_4}(x_4) \cdot \mu_{f_5\to x_4}(x_4) = \sum_{\sim x_1} f_1 f_3 f_6 \cdot \sum_{\sim x_4} f_2 \cdot f_5(x_4)$.

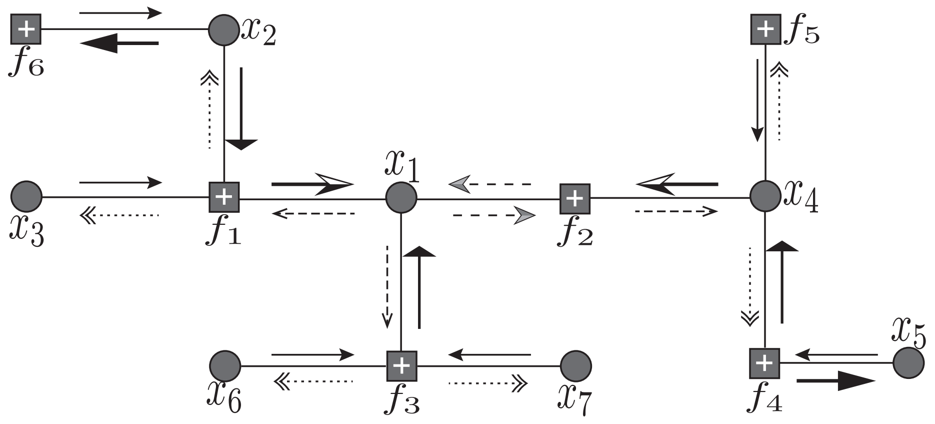


(f)

- **Round 7**

  The two remaining outgoing messages are:

  1. $\mu_{x_2 \to f_6}(x_2) = \mu_{f_1 \to x_2}(x_2) = \sum_{\sim x_2} f_1 f_2 f_3 f_4 f_5$
  2. $\mu_{f_4 \to x_5}(x_5) = \sum_{\sim x_5} f_4(x_4, x_5) \cdot \mu_{x_4 \to f_4}(x_4) = \sum_{\sim x_5} f_1 f_2 f_3 f_4 f_5 f_6$.



(g)

Figure 4.8: Implementation of the Belief Propagation Algorithm. Different arrows correspond to different rounds. Likewise, same arrows correspond to outgoing messages at the same round.

- **Final Marginalisation Step**

  In **Fig. 4.8(g)**, we observe that messages have been sent in both directions along every edge. Consequently, we will apply the bottom rule of **Fig. 4.7(b)** and we will have:

  1. $f_{X_1}(x_1) = \mu_{f_1\to x_1}(x_1)\mu_{f_3\to x_1}(x_1)\mu_{f_2\to x_1}(x_1) = \sum_{\sim x_1} f_1 f_2 f_3 f_4 f_5 f_6$.
  2. $f_{X_2}(x_2) = \mu_{f_1\to x_2}(x_2)\mu_{f_6\to x_2}(x_2) = \sum_{\sim x_2} f_1 f_2 f_3 f_4 f_5 f_6$.
  3. $f_{X_3}(x_3) = \mu_{f_1\to x_3}(x_3) = \sum_{\sim x_3} f_1 f_2 f_3 f_4 f_5 f_6$.
  4. $f_{X_4}(x_4) = \mu_{f_5\to x_4}(x_4)\mu_{f_4\to x_4}(x_4)\mu_{f_2\to x_4}(x_4) = \sum_{\sim x_4} f_1 f_2 f_3 f_4 f_5 f_6$.
  5. $f_{X_5}(x_5) = \mu_{f_4\to x_5}(x_5) = \sum_{\sim x_5} f_1 f_2 f_3 f_4 f_5 f_6$.
  6. $f_{X_6}(x_6) = \mu_{f_3\to x_6}(x_6) = \sum_{\sim x_6} f_1 f_2 f_3 f_4 f_5 f_6$.
  7. $f_{X_7}(x_7) = \mu_{f_3\to x_7}(x_7) = \sum_{\sim x_7} f_1 f_2 f_3 f_4 f_5 f_6$.

Therefore, applying the message-passing rules for BP algorithm, we compute all marginals efficiently on a single bidirectional tree. □

4.1.4 Decoding *via* Message Passing

Bit-Wise MAP Decoding

Assume a binary input $X_i \in \{\pm 1\}$ is transmitted over a memoryless channel, i.e., $p_{Y/X}(y/x) =$

$\prod_{i=1}^{n} p_{Y_i/X_i}(y_i/x_i)$, using a linear code $\mathcal{C}$ whose parity-check matrix $\mathcal{H}$ is defined in **4.4**. Also, assume that codewords are chosen uniformly at random i.e. $p(\mathbf{x}) = \frac{1}{|\mathcal{C}|}$. Then, the rule for the $bit-wise$ MAP decoder reads:

$$\begin{aligned}
\hat{x_i}^{MAP}(\mathbf{y}) &= \underset{x_i\in\{\pm1\}}{\text{argmax}}\; p_{X_i/\mathbf{Y}}(x_i/\mathbf{y}) \\
\text{(law of total probability)} &= \underset{x_i\in\{\pm1\}}{\text{argmax}} \sum_{\sim x_i} p_{\mathbf{X}/\mathbf{Y}}(\mathbf{x}/\mathbf{y}) \mathbb{1}_{\mathbf{x}\in\mathbf{C}(\mathbf{H})} \\
\text{(bayes's rule)} &= \underset{x_i\in\{\pm1\}}{\text{argmax}} \sum_{\sim x_i} p_{\mathbf{Y}/\mathbf{X}}(\mathbf{y}/\mathbf{x}) p_{\mathbf{X}}(\mathbf{x}) \mathbb{1}_{\mathbf{x}\in\mathbf{C}(\mathbf{H})} \\
\text{(memoryless channel, uniform priors)} &= \underset{x_i\in\{\pm1\}}{\text{argmax}} \sum_{\sim x_i} \left(\prod_{j} p_{Y_j/X_j}(y_j/x_j) \right) \mathbb{1}_{\mathbf{x}\in\mathbf{C}(\mathbf{H})} \qquad \mathbf{4.16}
\end{aligned}$$

Furthermore, using the factorisation in 4.6 and applying the rule of 4.16, bit-wise map decoder can be written as:

$$\hat{x_i}^{MAP}(\mathbf{y}) = \sum_{\sim x_i} \left(\prod_{j=1}^{5} p_{Y_j/X_j}(y_j/x_j) \right) \mathbb{1}_{\{x_3+x_4=0\}} \cdot \mathbb{1}_{\{x_1+x_2+x_3+x_5=0\}}. \qquad \mathbf{4.17}$$

Hence, from 4.17 it is clear that the bit-wise decoding problem is equivalent to calculating the marginal of a factorized function and choosing the value that maximizes this marginal. The corresponding factor graph of 4.17 is shown in Fig. 4.9. We have seen that factor nodes represent the *code membership* function. We set a function node connected with each variable node in order to represent the effect of the channel[2].

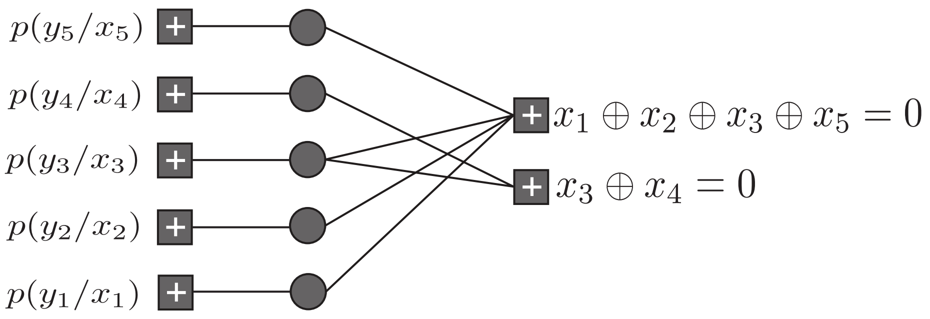


Figure 4.9: Extended factor graph of Fig. 4.2 containing the effect of the channel.

Notice that the above graph is a *tree*. We can therefore apply the message-passing algorithm to perform bit-wise MAP decoding.

Simplification of the Message-Passing rules

In the binary case (i.e., $X_i \in \{\pm 1\}$), a message $\mu(x)$ can be thought as a real-valued vector with length 2, $(\mu(1), \mu(-1))$. The initial such message sent from the factor leaf node to variable node i is $(p_{Y_i/X_i}(y_i/1), p_{Y_i/X_i}(y_i/-1))$. Recall that at a variable node of degree $K+1$ the message-passing rule is given by point-wise multiplication therefore,

$$\mu(1) = \prod_{k=1}^{K} \mu_k(1), \qquad \mu(-1) = \prod_{k=1}^{K} \mu_k(-1). \qquad \mathbf{4.18}$$

[2]$p(y_i/x_i)$ denotes the i-th channel realization to variable node i.

We introduce the ratio $r_k \triangleq \frac{\mu_k(1)}{\mu_k(-1)}$. These ratios are the likelihood ratios associated with channel observations. Consequently

$$r = \frac{\mu(1)}{\mu(-1)} = \frac{\prod_{k=1}^K \mu_k(1)}{\prod_{k=1}^K \mu_k(-1)} = \prod_{k=1}^K r_k. \tag{4.19}$$

That is, the ratio of the outgoing messages at a variable node is the product of the incoming ratios. Taking the log-likelihood ratios $l_k = \ln(r_k)$ we derive $l = \sum_{k=1}^K l_k$.

At check nodes (or factor nodes) the proof of the outgoing message ratio is more complex than the previous derivation. For a check node with degree $J+1$ the associated kernel function is

$$f(x, x_1, x_2, \cdots, x_J) = \mathbb{1}_{\{\prod_{j=1}^J x_j = x\}}. \tag{4.20}$$

Note that in **example 4.1** we had modulo-2 sum for the *code membership* function since we had bit values $\{0,1\}$. Now we consider bit values $\{\pm 1\}$, hence, the modulo-2 sum becomes product. Therefore

$$\begin{aligned} r = \frac{\mu(1)}{\mu(-1)} &= \frac{\sum_{\sim x} f(1, x_1, \cdots, x_J) \prod_{j=1}^J \mu_j(x_j)}{\sum_{\sim x} f(-1, x_1, \cdots, x_J) \prod_{j=1}^J \mu_j(x_j)} \overset{(2.16)}{=} \frac{\sum_{\sim x} \mathbb{1}_{\{\prod_{j=1}^J x_j = 1\}} \prod_{j=1}^J \mu_j(x_j)}{\sum_{\sim x} \mathbb{1}_{\{\prod_{j=1}^J x_j = -1\}} \prod_{j=1}^J \mu_j(x_j)} \\ &= \frac{\sum_{x_1,\cdots,x_J : \prod_{j=1}^J x_j = 1} \prod_{j=1}^J \mu_j(x_j)}{\sum_{x_1,\cdots,x_J : \prod_{j=-1}^J x_j = -1} \prod_{j=1}^J \mu_j(x_j)} = \frac{\sum_{x_1,\cdots,x_J : \prod_{j=1}^J x_j = 1} \prod_{j=1}^J \frac{\mu_j(x_j)}{\mu_j(-1)}}{\sum_{x_1,\cdots,x_J : \prod_{j=-1}^J x_j = -1} \prod_{j=1}^J \frac{\mu_j(x_j)}{\mu_j(-1)}} \\ &\overset{(a)}{=} \frac{\sum_{x_1,\cdots,x_J : \prod_{j=1}^J x_j = 1} \prod_{j=1}^J r_j^{\frac{(1+x_j)}{2}}}{\sum_{x_1,\cdots,x_J : \prod_{j=-1}^J x_j = -1} \prod_{j=1}^J r_j^{\frac{(1+x_j)}{2}}} \overset{(b)}{=} \frac{\prod_{j=1}^J (r_j + 1) + \prod_{j=1}^J (r_j - 1)}{\prod_{j=1}^J (r_j + 1) - \prod_{j=1}^J (r_j - 1)} \\ &\overset{(c)}{=} \frac{1 + \prod_j \frac{r_j - 1}{r_j + 1}}{1 - \prod_j \frac{r_j - 1}{r_j + 1}}. \end{aligned} \tag{4.21}$$

In (a) we simply define $r_j \triangleq \frac{\mu_j(x_j)}{\mu_j(-1)}$, which gives that $r_j^{\frac{(1+x_j)}{2}} = \begin{cases} 1, & x_j = -1, \\ r_j, & x_j = 1. \end{cases}$

In (b), we expand the products $\prod_{j=1}^J (r_j + 1)$, $\prod_{j=1}^J (r_j - 1)$ and we observe that

$$\prod_{j=1}^J (r_j + 1) + \prod_{j=1}^J (r_j - 1) = 2 \cdot \sum_{x_1,\cdots,x_J : \prod_{j=1}^J x_j = 1} \prod_{j=1}^J r_j^{\frac{(1+x_j)}{2}}.$$

Similarly,

$$\prod_{j=1}^{J}(r_j+1)-\prod_{j=1}^{J}(r_j-1)=2\cdot\sum_{x_1,\cdots,x_J:\prod_{j=1}^{J}x_j=-1}\prod_{j=1}^{J}r_j^{\frac{(1+x_j)}{2}}.$$

In (c) we divided numerator and denominator by $\prod_{j=1}^{J}(r_j+1)$.

Moreover

$$\frac{r-1}{r+1}=\frac{\frac{1+\prod_j\frac{r_j-1}{r_j+1}}{1-\prod_j\frac{r_j-1}{r_j+1}}-1}{\frac{1+\prod_j\frac{r_j-1}{r_j+1}}{1-\prod_j\frac{r_j-1}{r_j+1}}+1}=\frac{\frac{2\prod_j\frac{r_j-1}{r_j+1}}{1-\prod_j\frac{r_j-1}{r_j+1}}}{\frac{2}{1-\prod_j\frac{r_j-1}{r_j+1}}}=\prod_j\frac{r_j-1}{r_j+1}. \qquad \mathbf{4.22}$$

From $r=e^{\ell}$, it follows that

$$\frac{r-1}{r+1}=\tanh\left(\frac{\ell}{2}\right). \qquad \mathbf{4.23}$$

To verify **4.23**, we use the algebraic expression $tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$ and we take:

$$\tanh(\ell/2)=\frac{e^{\frac{\ell}{2}}-e^{-\frac{\ell}{2}}}{e^{\frac{\ell}{2}}+e^{-\frac{\ell}{2}}}=\frac{e^{-\frac{\ell}{2}}(e^{\ell}-1)}{e^{-\frac{\ell}{2}}(e^{\ell}+1)}=\frac{r-1}{r+1}.$$

Combining with the expression in **4.22** we have:

$$\frac{r-1}{r+1}=\tanh(\ell/2)=\prod_j\frac{r_j-1}{r_j+1}=\prod_{j=1}^{J}\tanh(\ell_j/2). \qquad \mathbf{4.24}$$

Hence,

$$\ell=2\cdot\tanh^{-1}\left(\prod_{j=1}^{J}\tanh(\ell_j/2)\right). \qquad \mathbf{4.25}$$

To summarize, in the case of transmission over a binary channel, the messages can be compressed to a single real quantity. More precisely, if we choose this quantity to be the log of the ratio of the two likelihoods, then the processing rules are simplified at variable nodes with the addition of individual messages and at check nodes the processing rule is stated in **4.25**.

4.1.5 BP decoding for the BEC

The *binary erasure channel* (BEC) shown in Figure 4.10 transmits one of two symbols, usually the binary digits $\{0,1\}$. However, the receiver either receives the bit correctly or it receives a message "?" that the bit was erased. In other words, BEC does not introduce errors. The BEC erases a bit with probability ϵ. Hence, the channel transition probabilities are:

$$p(Y=0/X=0)=1-\epsilon.$$
$$p(Y=0/X=1)=0.$$
$$p(Y=1/X=1)=1-\epsilon.$$
$$p(Y=1/X=0)=0.$$
$$p(Y=?/X=0)=\epsilon.$$
$$p(Y=?/X=1)=\epsilon.$$

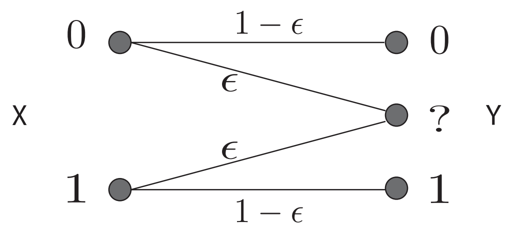


Figure 4.10: The binary erasure channel (BEC).

The BEC does not flip bits, therefore, if Y is received as 1 or 0 then the receiver can be completely certain of the value of X. Namely, $p(X=0/Y=0)=1$ and $p(X=1/Y=1)=1$.

In the previous subsections, we introduced a message-passing algorithm to accomplish the decoding task. We will now specialize this algorithm to the BEC. The Tanner graph of an LDPC code (and so the factor graph corresponding to the bit-wise MAP decoding) is not usually a tree. If the factor graph is a tree, then the decoding scheme is the classic message-passing procedure, i.e., start at the leaf nodes and send a message once all incoming messages required for the computation have arrived. Otherwise, (i.e. code with cycles) we proceed in iterations. We start by processing incoming messages at check nodes and then sending the resulting outgoing messages to variable nodes along all edges. These

messages are subsequently processed at the variable nodes and the outgoing messages are sent back along all edges to the check nodes. This constitutes one round of message passing. Note that, initially, variable nodes send the messages received from the channel to their neighbouring check nodes.
Hence the initial messages are:

$$\big(\mu_j(0), \mu_j(1)\big) = \big(p(y_j/0), p(y_j/1)\big) \qquad \textbf{4.26}$$

For the BEC, the possible initial messages are $\big(1-\epsilon, 0\big)$, $\big(\epsilon, \epsilon\big)$, $\big(0, 1-\epsilon\big)$ corresponding to the received values "0", "?" and "1" respectively. Recall that the normalization of the messages plays no role. We discussed in § 4.1.4, that we only need to know the ratio and this inference stays valid if the graph contains cycles. Therefore, equivalently we can work with the set of messages (1,0), (1,1) and (0,1).

We now state the processing rules. We claim that the general message-passing rules can be simplified to the following:
At a variable node, the outgoing message is an erasure if *all* incoming messages are erasures. Otherwise, namely if there is at least a non-erasure incoming message, the outgoing message is equal to this common value ("0" or "1"). This simplified rule can be proved easily due to message-passing rule in **Fig. 4.7(b)**. The outgoing message is computed by component-wise multiplication of all incoming messages. Consider a variable node with degree 3. If the two incoming messages are erasures, i.e., messages (1,1) and (1,1), then the outgoing message $\mu(x) = (1,1)\cdot(1,1) = (1,1)$ will be an erasure. Otherwise, if one of the two incoming messages is an erasure and the other is for instance the message "1" i.e. (0,1-ϵ) normalized (0,1) then, the outgoing message will be $\mu(x) = (1,1)\cdot(0,1) = (0,1)$-the common value of the incoming message. Hence, we confirmed the simplification rule.
At a check node, the outgoing message is an erasure if *any* of the incoming messages is an erasure. Otherwise, the outgoing message is the modulo-2 sum of the incoming messages. Consider again a check node with degree 3 with two incoming messages. Since check nodes of higher degree can be modelled as the cascade of several check nodes, each of which has two inputs and one output.
Assume that the incoming messages are $\big(\mu_1(0), \mu_1(1)\big) = (1,0)$ and $\big(\mu_2(0), \mu_2(1)\big) = (0,1)$. The message-passing rule at check node is stated in **Fig. 4.7(b)**. Moreover, due to **4.20**

the outgoing message is[3]

$$\begin{aligned}\big(\mu(0),\mu(1)\big) &= \left(\sum_{x_1,x_2} \mathbb{1}_{\{x_1+x_2=0\}}\mu_1(x_1)\mu_2(x_2), \sum_{x_1,x_2} \mathbb{1}_{\{x_1+x_2=1\}}\mu_1(x_1)\mu_2(x_2)\right)\\ &= \left(\sum_{x_1,x_2:x_1\oplus x_2=0} \mu_1(x_1)\mu_2(x_2), \sum_{x_1,x_2:x_1\oplus x_2=1} \mu_1(x_1)\mu_2(x_2)\right)\\ &= \big(\mu_1(1)\mu_2(1)+\mu_1(0)\mu_2(0), \mu_1(0)\mu_2(1)+\mu_1(1)\mu_2(0)\big)\\ &= \big(0\cdot 1+1\cdot 0, 1\cdot 1+0\cdot 0\big) = \big(0,1\big) \qquad \mathbf{4.27}\end{aligned}$$

Indeed, the result in **4.27** is equal to the modulo-2 sum of the incoming messages, since, $1 \oplus 0 = 1$ and (0,1) refer to "1" message. In addition, if the two incoming messages are $\big(\mu_1(0),\mu_1(1)\big) = (1,0)$ and $\big(\mu_2(0),\mu_2(1)\big) = (1,1)$, i.e., the second message is an erasure then, from **4.27**, the outgoing message will be equal to $(0\cdot 1+1\cdot 1, 1\cdot 1+0\cdot 1) = (1,1)$, i.e., an erasure as we expected.

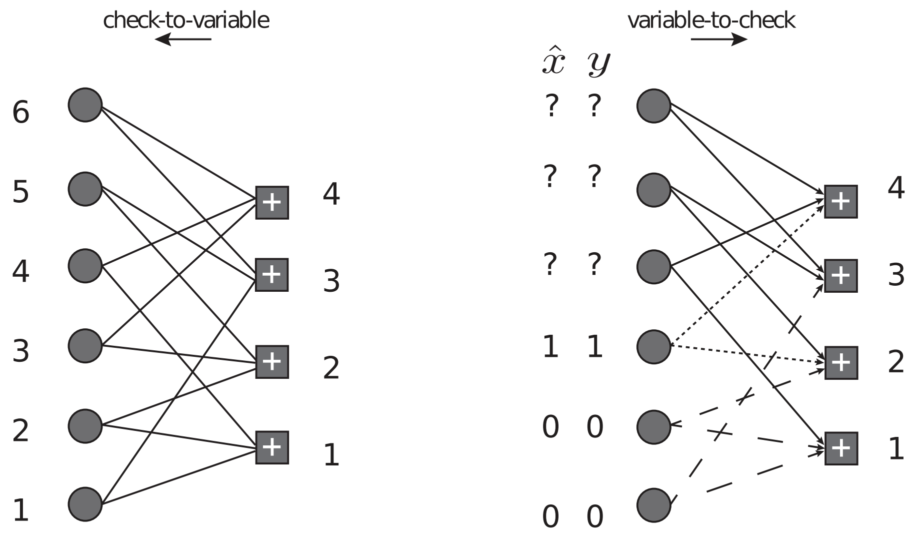
Example 4.3. Consider the LDPC code $\mathcal{C}$ which consists of all length-6 vectors $\mathbf{c} = (c_1\, c_2 \cdots c_6)$ with a regular parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 1 & 0 & 1 & 0\\ 1 & 0 & 0 & 0 & 1 & 1\\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Assume an encoded codeword $\mathbf{c} = [0\,0\,1\,0\,1\,1]$. The vector $\mathbf{c}$ is sent through the BEC and the vector $\mathbf{y} = [0\,0\,1\,?\,?\,?]$ is received. Message-passing decoding is used to recover the erased bits. Figure **4.11** shows graphically the messages passed in the message-passing decoder.

In iteration 0 (initialization step) the variable-to-check messages correspond to the received values. Consequently, $\hat{\mathbf{x}}$ which denotes the current estimate of the transmitted word is equal to $\mathbf{y}$.

[3]notice that in **4.20** there's a product. In our case, this will be replaced by a summation since we have bit values $\{0,1\}$ and not symbols ± 1.

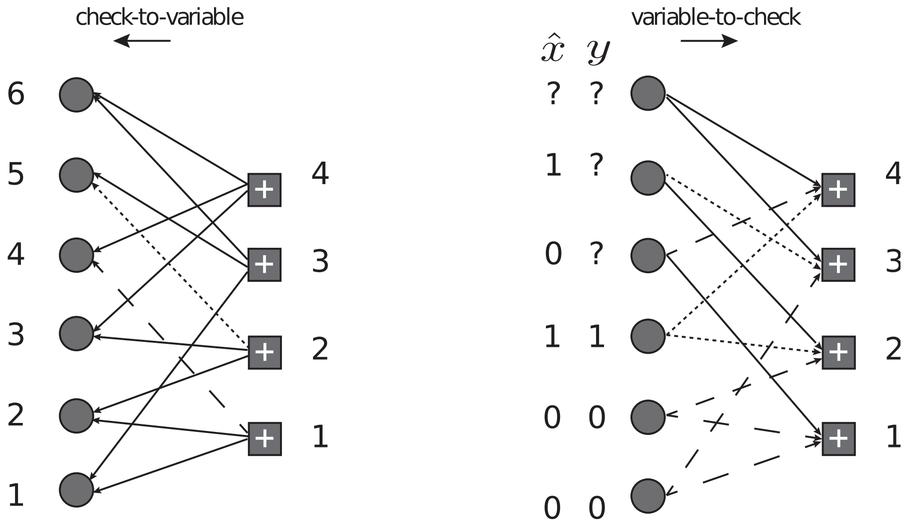


(a) Initialization step.

At iteration 1, the messages sent to variable nodes are shown on the left of Figure **4.11(b)**. For instance, the messages sent from check node 1 are

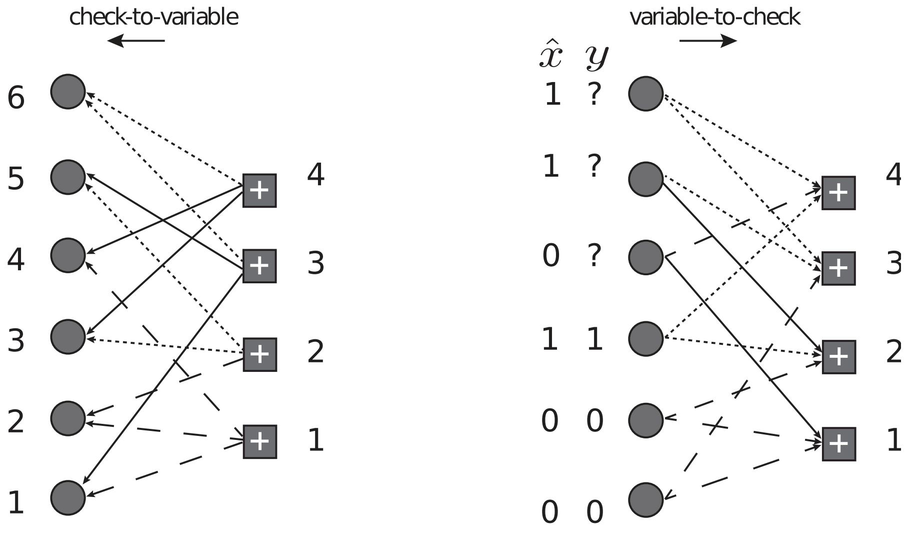
$$\mu_{\boxed{1}\to④}(y_4) = 0 \oplus 0 = 0, \qquad \mu_{\boxed{1}\to②}(y_2) = ?, \qquad \mu_{\boxed{1}\to①}(y_1) = ?\,.$$

We follow the same procedure for check nodes 2, 3 and 4. On the right of this figure there are the messages sent from variable to check nodes. Hence, the iteration 1 is completed. Observe that, in this round, the 4th variable node has the value 0 due to $\mu_{\boxed{1}\to④}(y_4)$ and the 5th has the value 1 due to message $\mu_{\boxed{2}\to⑤}(y_5) = 1 \oplus 0 = 1$. Therefore, we recovered the 4th and 5th bit.



(b) Iteration one.

At round 2, we attend the same as previously and we recover the 6th bit due to message $\mu_{\boxed{4}\to \textcircled{6}}(y_6) = 0 \oplus 1 = 1$.



(c) Iteration two.

Figure 4.11: Message passing decoding in order to find the erased part of the codeword. The dotted arrows correspond to messages "bit=1", the solid line correspond to message "bit= ?" and the broken arrows correspond to messages "bit=0".

Therefore, we recovered the codeword **c**. □

4.2 The principle of iterative decoding

Before studying methods of LDPC code analysis, we discuss the principle of iterative decoding [9] to make the goal of the analysis clear.

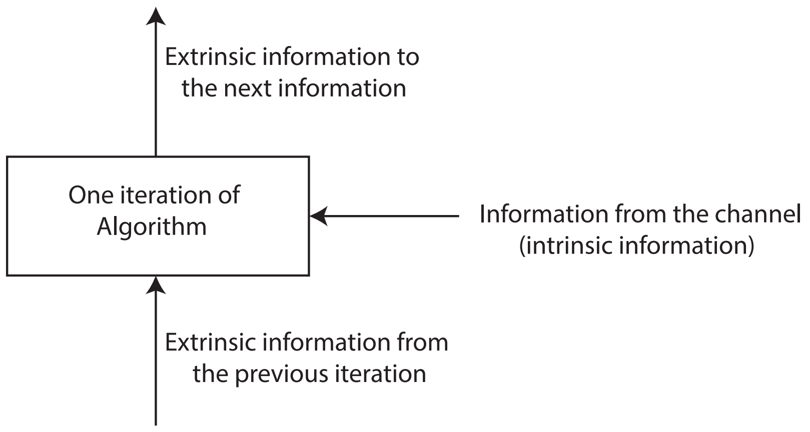

Extrinsic information to the next information
One iteration of Algorithm
Information from the channel (intrinsic information)
Extrinsic information from the previous iteration

Figure 4.12: The principle of iterative decoding.

An iterative decoder, as shown in **Fig. 4.12**, at each iteration uses two sources of knowledge about the transmitted codeword: the intrinsic information from the channel and the information from the previous iteration. From these sources, the decoding algorithm attempts to obtain a better knowledge about the transmitted codeword, using this knowledge as the extrinsic information for the next iteration. In a successful decoding, the extrinsic information is getting better and better as the decoder iterates. Therefore, in all methods of analysis of iterative decoders, the statistics of the extrinsic messages at each iteration are studied.

Studying the evolution of the pdf of extrinsic messages iteration by iteration is known as *density evolution*. However, as an approximate analysis, one may study the evolution of a representative of this density (e.g. mean of a Gaussian pdf).

4.3 The decoding tree

Consider an updated message from a variable node v of degree d_{v} to a check node in the decoder. This message is computed from $d_{\mathrm{v}} - 1$ incoming messages and the channel message. These $d_{\mathrm{v}} - 1$ incoming messages are in fact the outgoing messages of some check nodes, which are updated previously. Consider one of those messages with its check node c of degree d_{c}. The outgoing message of this check node is computed from $d_c - 1$ incoming messages to c. One can repeat this for all the check nodes connected to v to form a decoding tree of depth one. An example of such a decoding tree is depicted in **Fig. 4.13**.

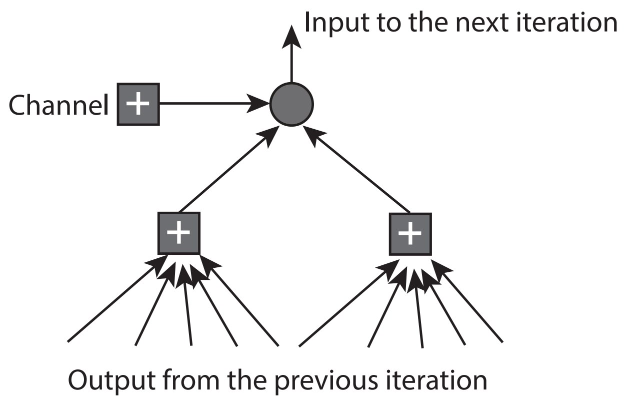

Input to the next iteration
Channel
Output from the previous iteration

Figure 4.13: The depth-one decoding tree for a regular (3,6) LDPC code.

Continuing in the same fashion, one can get the decoding tree of any depth. **Fig. 4.14** shows an example of a depth-two decoding tree. Notice that, when the factor graph is a *tree*, the messages in the decoding tree of any depth are *indepedent*. If the factor graph has cycles and its girth is $2 \cdot \ell$, then up to depth ℓ the messages in the decoding tree are *indepedent*. Therefore, the independence assumption is correct up to ℓ iterations and is an approximation for subsequent iterations.

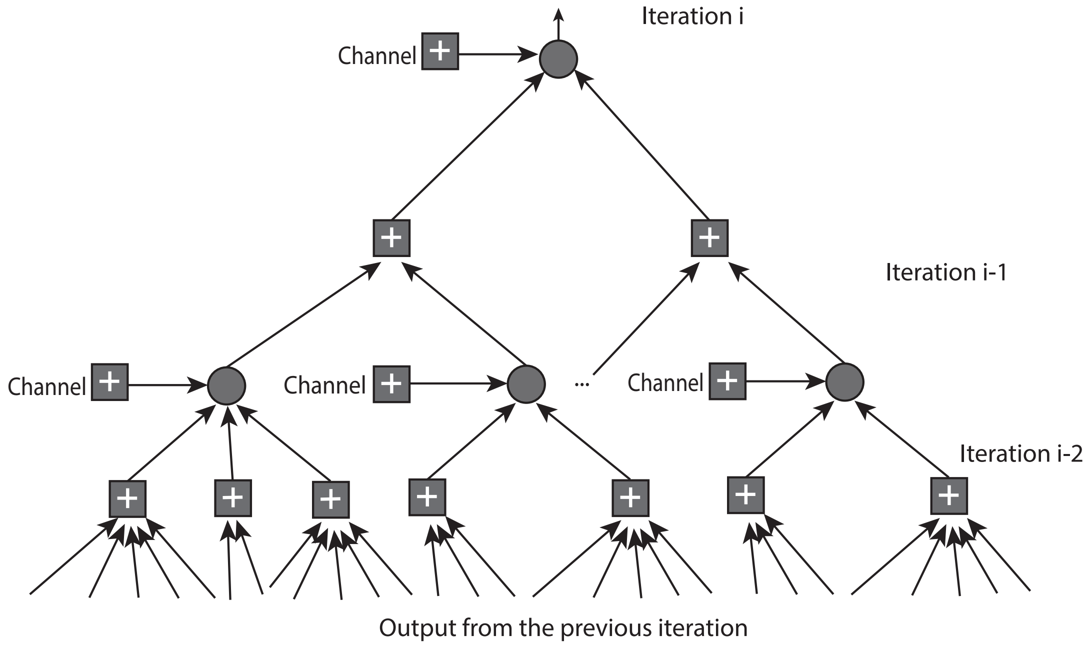


Figure 4.14: The depth-two decoding tree for an irregular LDPC code.

Chapter 5

Analysis & Design of LDPC codes

5.1 Density Evolution for LDPC Codes

In 2001, Richardson and Urbanke [5] proposed a technique called *density evolution*, which tracks the evolution of the pdf of the messages, iteration by iteration. Density Evolution is the most common method for asymptotic behaviour analysis. It can predict the decoding performance of a code ensemble in the limit of infinite blocklength using only the degree distributions of the code. To be able to define a density for the messages, they needed a property for channel and decoding, called the *symmetry conditions*. The symmetry conditions require the channel to be *symmetric* (see Definition in **2.2**) and the decoding update rules to satisfy some symmetry properties as follows.

Check node symmetry: The check node update rule [9, p.25] is symmetric if

$$\mathrm{CHK}(b_1 m_1,\, b_2 m_2, \dots,\, b_{d_c-1} m_{d_c-1}) = \mathrm{CHK}(m_1,\, m_2, \dots,\, m_{d_c-1})\left(\prod_{i=1}^{d_c-1} b_i\right) \qquad \textbf{5.1}$$

for any ± 1 sequence $(b_1, b_2, \dots, b_{d_c-1})$. Namely, signs factor out of check node messages maps. Here, CHK() is the check update rule, which takes $d_c - 1$ input messages and generates one output message.

Variable node symmetry: The variable node update rule [9, p.25] is symmetric if

$$\mathrm{VAR}(-m_0, , -m_1, \dots, -m_{d_v-1}) = -\mathrm{VAR}(m_0, , m_1, \dots, m_{d_v-1}) \qquad \textbf{5.2}$$

i.e. sign inversion invariance of variable node message maps holds. Here, VAR() is the variable node update rule, which has inputs $d_\text{v} - 1$ messages together with the channel message m_0 and generates one output message.

Under the symmetry conditions and assuming a linear code, the convergence behaviour of the decoder is independent of the transmitted codeword. Consequently, one may assume that the all-zero codeword is transmitted. In practice, *discrete density evolution* [11] is used. The idea is to quantize the message alphabet and use pmfs instead of pdfs to make a computer implementation possible.
Density evolution is not specific to LDPC codes. It is a technique which can be adopted for other codes defined on graphs associated with iterative decoding. However, it becomes intractable when the codes are complex as in turbo codes [9, p.25].

5.1.1 BEC Channel

Rather than analysing individual codes it suffices to assess the *ensemble average performance*. This is true, since, as the next theorem asserts, the individual elements of an ensemble behave with high probability close to the ensemble average.

Theorem 5.1. ([1, p.85], Concentration around Ensemble Average). Let a Tanner graph G, chosen uniformly at random from LDPC(n, λ, p), be used for transmission over the BEC(ϵ). Assume that the decoder performs ℓ rounds of message-passing decoding and let $P_b^{BP}(G, \epsilon, \ell)$ denote the resulting bit erasure probability. Then for ℓ fixed and for any given $\delta > 0$, there exist an $\alpha > 0$, $\alpha = \alpha(\lambda, p, \epsilon, \delta, \ell)$ such that

$$P\Big\{ \Big| P_b^{BP}(G, \epsilon, \ell) - \mathbb{E}_{G' \in LDPC(n,\lambda,p)}\big[P_b^{BP}(G', \epsilon, \ell) \big] \Big| > \delta \Big\} \leq e^{-\alpha \cdot n}. \qquad \textbf{5.3}$$

This probability tends to 0 as the blocklength tends to infinity, since δ is strictly positive.

In words, the theorem asserts that all except an exponentially (in the blocklength) small fraction of codes behave within an arbitrarily small δ from the ensemble average. Assuming sufficiently large blocklengths, the ensemble average behaviour is a good indicator for the individual behaviour. We therefore focus our effort on the design and construction of ensembles whose average performance approaches the Shannon theoretic limit.

The concentration results proved in [5] guarantee that, for sufficiently large block lengths, almost every code in the given ensemble will have vanishing probability of bit error for channels with parameters below the calculated threshold.

A first big simplification stems from the realization that the performance is independent of the transmitted codeword and is only a function of the erasure pattern: at any iteration, the set of known variable nodes is only a function of the set of known messages but independent of their values. The equivalent statement is true for the set of known check nodes.

Theorem 5.2. ([1, p.85], Conditional Independence of Erasure Probability). Let G be a Tanner graph representing a binary linear code $\mathcal{C}$. Assume that $\mathcal{C}$ is used to transmit over BEC(ϵ) and assume that the decoder performs message-passing decoding on G. Let $P_b^{BP}(G,\epsilon,\ell,\mathbf{x})$ denote the conditional bit erasure probability after the ℓ-th decoding iteration assuming $\mathbf{x}$ was sent, $\mathbf{x}\in\mathcal{C}$. Then,

$$P_b^{BP}(G,\epsilon,\ell,\mathbf{x}) = \frac{1}{|\mathcal{C}|}\sum_{\mathbf{x}'\in\mathcal{C}} P_b^{BP}(G,\epsilon,\ell,\mathbf{x}') = P_b^{BP}(G,\epsilon,\ell) \qquad \mathbf{5.4}$$

i.e. $P_b^{BP}(G,\epsilon,\ell,\mathbf{x})$ is independent of the transmitted codeword.

As a consequence, we are free to choose a particular codeword and to analyse the performance of the system assuming that this codeword was sent. It is natural to assume that the “all-zero word”, which is contained in every linear code, was sent. We refer to this assumption as the “all-zero codeword” assumption.

Another important statement is that for long codes (i.e. large block lengths) the average behaviour *converges* (in the block length n) to that of the cycle-free case.

Description of Density Evolution

The basic assumption is that the computation graph [1, p.87] spanned by belief-propagation at round ℓ (denoted $\mathcal{T}_\ell$) is a tree, meaning that the messages exchanged by the BP algorithm are independent, hence all APPs are marginalised correctly. In BEC case, Density Evolution tracks the mean message erasure probability of the variable-to-check messages for each round of the belief propagation algorithm. By definition, the initial messages to check nodes are the observed values of the variable nodes, which have an erasure probability $\epsilon\in[0,1]$.

Theorem 5.3. ([1, p.95]). Consider a degree distribution pair (λ,ρ). Then there is an explicit recursion which connects the distributions of messages passed from variable nodes to check nodes at two consecutive rounds of belief-propagation. The expected fraction of erasure messages which are passed in the ℓ-th iteration, call it x_ℓ, evolves as

$$x_\ell = \epsilon\cdot\lambda(1-\rho(1-x_{\ell-1})). \qquad \mathbf{5.5}$$

Proof. At round 0 the mean variable-to-check erasure probability under BP is $P_{\mathcal{T}_0}^{BP} = \epsilon$, where $P_{\mathcal{T}_\ell}^{BP}$ denotes the mean variable-to-check erasure probability under BP at round ℓ. Let $P_{\mathcal{T}_\ell}^{BP} = x_\ell$.We will now use induction to find an expression for x_ℓ as a function of the erasure probability at the previous iteration, namely $x_{\ell-1}$. We start with check-to-variable messages in the $(\ell+1)$-th round. Recall that by definition of the algorithm a check-to-variable message emitted by a check node of degree i along a particular edge is the erasure message, if any of the $i-1$ incoming messages is an erasure. By assumption, each such message is an erasure with probability x_ℓ and all messages are independent, so that the probability that the outgoing message is an erasure is equal $1-(1-x_\ell)^{i-1}$. Since the edge has probability ρ_i to be connected to a check node of degree i it follows that the expected erasure probability of a check-to-variable message at $(\ell+1)$-th round is equal to $\sum_i \rho_i \cdot (1-(1-x_\ell)^{i-1}) = 1-\rho(1-x_\ell)$. Now consider the erasure probability of the variable-to-check messages in the $(\ell+1)$-th round. Consider an edge $\mathbf{e}$ that is connected to a variable node of degree i. The outgoing variable-to-check message along this edge in the $(\ell+1)$-th round is an erasure if the received value of the associated variable node is an erasure and all $i-1$ incoming messages are erasures. This happens with probability $\epsilon \cdot (1-\rho(1-x_\ell))^{i-1}$. Averaging again over the edge degree distribution λ we get that $P_{\mathcal{T}_{\ell+1}}^{BP} = x_{\ell+1} = \sum_i \lambda_i \cdot \epsilon \cdot (1-\rho(1-x_\ell))^{i-1} = \epsilon \cdot \lambda(1-\rho(1-x_\ell))$ as claimed. ■

We now proceed to the introduction of some useful properties of density evolution, namely the *monotonicity* (with respect to the channel and the iteration number ℓ), the concept of the *threshold* and, finally, the *stability condition* [1].

Monotonicity

Monotonicity either with respect to the channel parameter or with respect to the number of iterations ℓ plays a fundamental role in the analysis of density evolution. The first lemma is a direct consequence of the non-negativity of the coefficients of the polynomials $\lambda(x)$ and $\rho(x)$ and the fact that $\rho(1) = 1$. We skip the proof.

Lemma 5.1. ([1, p.96], Monotonicity of $f(\cdot,\cdot)$). For a given distribution pair (λ, ρ) define $f(\epsilon, x) = \epsilon \cdot \lambda(1-\rho(1-x))$. Then $f(\epsilon, x)$ is increasing in both its argument for $x, \epsilon \in [0,1]$.

Lemma 5.2. ([1, p.96], Monotonicity with respect to channel). Let (λ, ρ) be a degree distribution pair and $\epsilon \in [0,1]$. If $P_{\mathcal{T}_\ell}^{BP}(\epsilon) \overset{\ell\to\infty}{\longrightarrow} 0$ then $P_{\mathcal{T}_\ell}^{BP}(\epsilon') \overset{\ell\to\infty}{\longrightarrow} 0$ for all $0 \le \epsilon' \le \epsilon$.

Proof. Recall from Theorem 5.3 that $P^{BP}_{\mathcal{T}_\ell}(\epsilon) = x_\ell(\epsilon)$, where $x_0(\epsilon) = \epsilon$, $x_\ell(\epsilon) = f(\epsilon, x_{\ell-1}(\epsilon))$ and $f(\epsilon, x) = \epsilon \cdot \lambda(1-\rho(1-x))$. Assume that for some $\ell \geq 0$, $x_\ell(\epsilon^{'}) \leq x_\ell(\epsilon)$. Then

$$x_{\ell+1}(\epsilon^{'}) = f(\epsilon^{'}, x_\ell(\epsilon^{'})) \overset{Lem.5.1}{\leq} f(\epsilon, x_\ell(\epsilon)) = x_{\ell+1}(\epsilon).$$

But if $\epsilon^{'} \leq \epsilon$, then $x_0(\epsilon^{'}) \leq x_0(\epsilon)$ and we conclude by induction that $x_\ell(\epsilon^{'}) \leq x_\ell(\epsilon)$. So if $x_\ell(\epsilon) \overset{\ell\to\infty}{\longrightarrow} 0$ then $x_\ell(\epsilon^{'}) \overset{\ell\to\infty}{\longrightarrow} 0$. ■

Lemma 5.3. ([1, p.97], Monotonicity with respect to iteration). Let $\epsilon, x_0 \in [0,1]$. For $\ell = 1, 2, \cdots$ define $x_\ell(x_0) = f(\epsilon, x_{\ell-1}(x_0))$. Then $x_\ell(x_0)$ is a monotone sequence converging to the nearest (in the direction of monotonicity) solution of the equation $f(\epsilon, x) = x$.

Proof. If $x_0 = 0$ or $\epsilon = 0$ then $x_\ell = 0$ for $\ell \geq 1$ and the fixed point is 0. If for some $\ell \geq 1$, $x_\ell \geq x_{\ell-1}$ then $x_{\ell+1} = f(\epsilon, x_\ell) \overset{Lem.5.1}{\geq} f(\epsilon, x_{\ell-1}) = x_\ell$ and the corresponding conclusion holds if $x_\ell \leq x_{\ell-1}$. This proves the monotonicity of sequence $\{x_\ell\}_{\ell\geq 0}$. Since for $\epsilon \geq 0$ we have $0 \leq f(\epsilon, x) \leq \epsilon$ for all $x \in [0,1]$, it follows that x_ℓ converges to an element of $[0,\epsilon]$-call it x_∞. It remains to show that x_∞ is the nearest fixed point. Consider a fixed point z such that $x_\ell(x_0) \leq z$ for some $\ell \geq 0$. Then $x_{\ell+1}(x_0) = f(\epsilon, x_\ell(x_0)) \overset{Lem.5.1}{\leq} f(\epsilon, z) = z$ which shows that $x_\infty \leq z$. Similarly, if $x_\ell(x_0) \geq z$ then $x_\infty \geq z$. This shows that x_ℓ cannot "jump" over any fixed point and must therefore converge to the nearest one. ■

Threshold

From the density evolution equations, we observe that, for every non-negative integer ℓ, $P^{BP}_{\mathcal{T}_\ell}(\epsilon = 0) = 0$ and $P^{BP}_{\mathcal{T}_\ell}(\epsilon = 1) = 1$ and these equalities hold for $\ell \to \infty$. Hence, a supremum of ϵ for which $P^{BP}_{\mathcal{T}_\ell} \overset{\ell\to\infty}{\longrightarrow} 0$ must exist. This value is called the *threshold*.

Definition 5.1. ([1, p.97]). The *threshold* associated with the degree distribution pair (λ, ρ) call it $\epsilon^{BP}(\lambda, \rho)$, is defined as

$$\epsilon^{BP}(\lambda, \rho) = \sup\{\epsilon \in [0,1] : P^{BP}_{\mathcal{T}_\ell(\lambda,\rho)}(\epsilon) \overset{\ell\to\infty}{\longrightarrow} 0\}. \qquad \mathbf{5.6}$$

◇

The preceding definition of the threshold is not very convenient for the purpose of analysis. We therefore state a second equivalent definition based on the fixed points of density evolution.

Theorem 5.4. **([1, p.98],).** For a given degree distribution pair (λ, ρ) and $\epsilon \in [0,1]$ let $f(\epsilon, x) = \epsilon \cdot \lambda(1 - \rho(1-x))$.

(i). $\epsilon^{BP}(\lambda, \rho) = \sup\{\epsilon \in [0,1] : x = f(\epsilon, x)\,\text{has no solution}\, x \text{ in } (0,1]\}$.

(ii). $\epsilon^{BP}(\lambda, \rho) = \inf\{\epsilon \in [0,1] : x = f(\epsilon, x)\,\text{has a solution}\, x \text{ in } (0,1]\}$.

Proof. Let $x(\epsilon)$ be the largest solution in $[0,1]$ to $x = f(\epsilon, x)$. Note that for any $x \in [0,1]$ we have $0 \le f(\epsilon, x) \le \epsilon$. We conclude that $x(\epsilon) \in [0, \epsilon]$. By Lemma 5.3, we have $x_\ell(\epsilon) \overset{\ell \to \infty}{\longrightarrow} x(\epsilon)$. We conclude that if $x(\epsilon) > 0$ then ϵ is above the threshold, whereas if $x(\epsilon) = 0$ then ϵ is below the threshold. ■

The fixed point characterisation gives rise to the following convenient graphical method for determining the *threshold*. Draw $f(\epsilon, x) - x$ as a function of x, $x \in (0,1]$. The *threshold* ϵ^{BP} is the largest ϵ such that the graph of $f(\epsilon, x) - x$ is negative.

Stability condition

Expanding the right-hand size of 5.5 into a Taylor series around zero we get

$$x_\ell = \epsilon \lambda'(0) \rho'(1) x_{\ell-1} + \mathcal{O}(x_{\ell-1}^2). \qquad \textbf{5.7}$$

For sufficiently small x_ℓ the convergence behaviour is determined by the term linear in x_ℓ. More precisely, the convergence depends on whether $\epsilon \lambda'(0) \rho'(1)$ is smaller or larger than 1 [1, p.100].

Theorem 5.5. **([1, p.100]).** Assume that we are given a degree distribution pair (λ, ρ) and $\epsilon, x_0 \in [0,1]$. Let $x_\ell(x_0) = f(\epsilon, x_{\ell-1}(x_0))$. Then

(i). (Necessity) If $\epsilon \lambda'(0) \rho'(1) > 1$ then there exists a strictly positive constant $\xi = \xi(\lambda, \rho, \epsilon)$ such that $\lim_{\ell \to \infty} x_\ell(x_0) \ge \xi$ for all $x_0 \in (0,1)$.

(ii). (Sufficiency) If $\epsilon \lambda'(0) \rho'(1) < 1$ then there exists a strictly positive constant $\xi = \xi(\lambda, \rho, \epsilon)$ such that $\lim_{\ell \to \infty} x_\ell(x_0) = 0$ for all $x_0 \in (0, \xi)$.

Notice that $f(\epsilon, 0) = 0$ for any initial ϵ, hence zero is a *fixed point* of the recursion in 5.5. Therefore, the preceding condition is the *stability condition* of the fixed point at zero.

The most important consequence of the *stability condition* is the implied upper bound on the threshold:

$$\epsilon^{BP}(\lambda,\rho) \leq \frac{1}{\lambda'(0)\rho'(1)}.$$

This concludes our quick analysis of density evolution for the BEC. We will now proceed to the more general family of Binary Memoryless Symmetric-Output Channels (BMSC), of which the BEC is a member as well.

5.1.2 BMS Channels

For general binary-input memoryless output-symmetric channels, the situation is much more involved since one has to keep track of the evolution of general distributions, which usually cannot be parameterized by a single parameter. Let us begin by recalling the *belief-propagation* algorithm. We will use the standard binary PAM map $0 \to +1$, $1 \to -1$ throughout. At each iteration, messages are passed along the edges of the graph from variable nodes to their incident check nodes and back. The messages are typically real-valued but they can also take on the values $\pm\infty$, reflecting the situation where some bits are known with absolute certainty.

Generically, messages which are sent in the ℓ-th iteration will be denoted by $m^{(\ell)}$. By $m_{\mathrm{vc}}^{(\ell)}$ we denote the message sent from variable node v to its incident check node c, while by $m_{\mathrm{cv}}^{(\ell)}$ we denote the message passed from check node c to its incident variable node v. Each message is in log-likelihood ratio form where X is the random variable describing the codeword bit value associated to variable node v, and $\mathbf{y}$ is the random variable describing all the information incorporated into this message. By Bayes rule, we have

$$m = \ln\frac{p(X=1/\mathbf{y})}{p(X=-1/\mathbf{y})} = \ln\frac{p(\mathbf{y}/X=1)}{p(\mathbf{y}/X=-1)} \qquad \mathbf{5.8}$$

since X is equally likely ± 1. The message m is the log-likelihood ratio of the random variable X under the *independence assumption.*

Let $m_{\mathrm{vc}}^{(\ell)}$ be a message from a variable node to a check node. Under BP decoding, $m_{\mathrm{vc}}^{(\ell)}$ is equal to the sum of all incoming LLRs, i.e.,

$$m_{\mathrm{vc}}^{(\ell)} = \sum_{i=0}^{d_{\mathrm{v}}-1} m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}\,, \qquad \mathbf{5.9}$$

where $m_{\mathrm{c}_i\mathrm{v}}$, $i = 1, \cdots, d_{\mathrm{v}} - 1$, are the incoming LLRs from the neighbours of the variable node except for the check node that gets the message $m_{\mathrm{vc}}^{(\ell)}$, and $m_{\mathrm{c}_0\mathrm{v}}^{(\ell)}$ is the channel observation. Since $m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}$ are independent, the density $f_{m_{\mathrm{vc}}^{(\ell)}}$ of $m_{\mathrm{vc}}^{(\ell)}$ is the convolution of the densities of the $m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}$ which can be efficiently computed in Fourier domain. Let $f_{m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}}$ denote the density of a random variable $m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}$, then

$$f_{m_{\mathrm{vc}}^{(\ell)}} = \bigotimes_{i=0}^{d_{\mathrm{v}}-1} f_{m_{\mathrm{c}_i\mathrm{v}}^{(\ell)}}, \qquad \mathbf{5.10}$$

where $\otimes$ denotes convolution.

Before continuing, we need to write the check node message update rule in a more convenient form. Recall that we can restrict ourselves to the all-zero codeword for analysis, which in this case becomes the all-one codeword, according to the BPSK mapping. Then, the distribution of the log-likelihood ratio $\ell(Y)$ associated with a channel observation Y assuming that $X = 1$, is denoted with $\mathcal{L}$ and is called an L-density. Some L-densities of commonly used channel models [1] are:

$$\begin{aligned}
\mathcal{L}_{\mathrm{BEC}(\epsilon)}(y) &= \epsilon\Delta_0(y) + (1-\epsilon)\Delta_{+\infty}(y),\\
\mathcal{L}_{\mathrm{BSC}(\epsilon)}(y) &= \epsilon\Delta_{-\ln\frac{1-\epsilon}{\epsilon}}(y) + (1-\epsilon)\Delta_{\ln\frac{1-\epsilon}{\epsilon}}(y),\\
\mathcal{L}_{\mathrm{BI\text{-}AWGN}(\sigma)}(y) &= \sqrt{\frac{\sigma^2}{8\pi}}\exp\left\{-\frac{(y-\frac{2}{\sigma^2})^2\sigma^2}{8}\right\}.
\end{aligned}$$

where $\Delta_z(x) = \Delta_0(x-z)$ and $\Delta_0(x)$ is the Dirac delta centered at zero.

A function which is closely related to $\ell(y)$ is:

$$d(y) = \tanh\left(\frac{\ell(y)}{2}\right) = \frac{1-e^{-\ell(y)}}{1+e^{-\ell(y)}} = p(X = 1/y) - p(X = -1/y),$$

where in the last step we have used Bayes rule and assumed that $p_X(1) = p_X(-1) = \frac{1}{2}$. It is easy to see that $d(y)$ takes values in the interval $[-1, 1]$. Also, we observe that from $\ell(y)$ we can compute $d(y)$, and vice versa. When we conceive $d(y)$ as a random variable we write $D = d(y)$. Conditioned on $X = 1$, we have $d(y) \in (-1, 1]$. The distribution of D conditioned on $X = 1$ is termed a D-distribution and is denoted by $\mathcal{D}$. If α is a D-density the symmetry takes the form

$$\frac{\alpha(y)}{\alpha(-y)} = \frac{1+y}{1-y}.$$

We remind that the symmetry for an L-density is given in **2.10**.
Another important quantity is

$$g(y) \triangleq (\operatorname{sign}(y), \ln\coth(|\ell(y)|/2)) = (\operatorname{sign}(d(y)), -\ln|d(y)|). \qquad \mathbf{5.11}$$

Again when we conceive $g(y)$ as a a random variable, we write $G = g(y)$. The density of G, conditioned on $X = 1$, is called a G-density and is denoted by $\mathcal{G}$. Also, note that we define the sign(y) as 1 if $y \geq 0$ and -1 otherwise. In addition, $g(y)$ takes values in $\{\pm1\} \times [0, +\infty]$. A G-density $\alpha(s, x)$ therefore has the form

$$\alpha(s, x) = \mathbb{1}_{s=1}\alpha(1, x) + \mathbb{1}_{s=-1}\alpha(-1, x).$$

A symmetric G-density exhibits the form

$$\alpha(1, x) = \alpha(-1, x)\coth(x/2).$$

We will see that the convolution of L-densities represents the message distribution change at variable nodes, and the convolution of G-densities represents the message distribution change at check nodes. L-densities, are real-valued functions, hence, their convolution denoted by $\otimes$ is well defined. However, G-densities take values in $\{\pm1\} \times [0, +\infty]$, hence, calculating their convolution is not trivial. The space $\mathbb{R}^+$ has the well-defined convolution. Instead of $\{\pm1\}$ we think the set $\{0, 1\}$ with modulo-two addition. The associated convolution is the cyclic convolution of sequences of length two. Therefore, the convolution of G-densities is just the two-dimensional convolution which consists of the familiar convolution over $\mathbb{R}^+$ in one dimension and the convolution over $\{0, 1\}$ in the other dimension. In other words, the new convolution is a convolution over the group $\{\pm1\} \times [0, +\infty]$.
For notation purposes, the convolution in G-domain will be denoted by $\star$. Therefore, if a and b are L-densities, then by writing $a \star b$ we refer to the transformation of a and b into G-densities, their convolution in the G-domain and their transformation back to the L-domain.

Description of Density Evolution

We remind that the processing rule at check nodes is given by

$$m_{\mathrm{cv}}^{(\ell)} = 2\tanh^{-1}\left(\prod_{i=1}^{d_{\mathrm{c}}-1} \tanh\left(\frac{m_{\mathrm{v}_i\mathrm{c}}^{(\ell)}}{2}\right)\right), \qquad \mathbf{5.12}$$

where $m_{\mathrm{v}_i c}^{(\ell)}$, $i = 1, \cdots, d_c - 1$ are the incoming LLRs from $d_c - 1$ neighbours of a check node, and $m_{\mathrm{cv}}^{(\ell)}$ is the message sent to the remaining neighbour.

For determining the density of $m_{\mathrm{cv}}^{(\ell)}$, we need to write the check node message update rule in a more convenient form. According to [6], we define a map $\gamma : [-\infty, +\infty] \to \{0,1\} \times [0, +\infty]$ as follows. Given $x \in [-\infty, +\infty]$, $x \neq 0$, let

$$\gamma(x) = \Big(\gamma_1(x), \gamma_2(x)\Big) \triangleq \Big(\operatorname{sgn}(x), -\ln\tanh\left|\frac{x}{2}\right|\Big). \qquad \textbf{5.13}$$

We define $-\ln(0) = +\infty$. Notice that this map is the G-density defined previously. Hence, $\operatorname{sgn}(x)$ is 1 if $x \geq 0$ and -1 otherwise.

Using $\gamma(x)$, the check node message update rule can be rewritten as

$$m_{\mathrm{cv}}^{(\ell)} = \gamma^{-1}\left(\sum_{i=1}^{d_c-1} \gamma\Big(m_{\mathrm{v}_i c}^{(\ell)}\Big)\right), \qquad \textbf{5.14}$$

where addition is performed component-wise. This form is more convenient because, given the *indepedence assumption* and the densities of all $m_{\mathrm{v}_i c}^{(\ell)}$, it is relatively easy to calculate the density of $m_{\mathrm{cv}}^{(\ell)}$. Let Γ denote the density transformation corresponding to $\gamma(x)$, i.e., given random variable $Z \in \mathbb{R}$ with density $f_Z(z)$, the density of $\gamma(Z)$ is

$$f_{\gamma(Z)}(\gamma_1(z), \gamma_2(z)) = \Gamma(f_Z(z)).$$

The aforementioned transformation has a well defined inverse, denoted by Γ^{-1}, and both Γ and Γ^{-1} are additive operators on their respective domain spaces [6]. The range space of Γ is endowed with a convolution operator $\star$. Since incoming LLRs at a check node are independent, the density $f_{m_{\mathrm{cv}}^{(\ell)}}$ of $m_{\mathrm{cv}}^{(\ell)}$ can be computed as

$$f_{m_{\mathrm{cv}}^{(\ell)}} = \Gamma^{-1}\Big(\Gamma\Big(f^{(\ell)}_{m_{\mathrm{v}_1 c}}\Big) \star \cdots \star \Gamma\Big(f^{(\ell)}_{m_{\mathrm{v}_{d_c-1} c}}\Big)\Big). \qquad \textbf{5.15}$$

More precisely, let P_ℓ and Q_ℓ denote the shorthand notations for the densities of random variables $m_{\mathrm{vc}}^{(\ell)}$ and $m_{\mathrm{cv}}^{(\ell)}$ respectively. By **5.14**, we see that the random variable describing the message passed from check node c to variable node v is the image under γ^{-1} of a sum of random variables from $\{0,1\} \times [0, +\infty]$. As mentioned earlier, these random variables are independent by the *independence assumption*. So, the density of their sum is the convolution of their densities.

We wish to calculate the density of the messages emanating from a check node of degree i, denoted $Q_{\ell,i}$. At a check node of degree i, $(i-1)$ incoming messages are summed after passing through the transformation $\gamma(x)$. The result is then transformed into L-domain using Γ^{-1}. All incoming messages are independent and distributed according to P_ℓ and their images under $\gamma(x)$ are also independent and distributed according to $\Gamma(P_\ell)$. Hence, the density of the messages emanating from a check node of degree i is given by

$$Q_{\ell,i} = \Gamma^{-1}\Big(\Gamma(P_\ell)^{\star(i-1)}\Big). \tag{5.16}$$

Summing up over all the possibilities for the degrees of the check node c we lead to

$$Q_\ell = \sum_{\geq 2} \rho_i Q_{\ell,i} = \sum_{i\geq 2} \rho_i \cdot \Gamma^{-1}\Big(\Gamma(P_\ell)^{\star(i-1)}\Big). \tag{5.17}$$

Since, Γ^{-1} is additive then

$$Q_\ell = \Gamma^{-1}\left(\sum_{i\geq 2} \rho_i \Gamma(P_\ell)^{\star(i-1)}\right) \tag{5.18}$$

and with a slightly abuse of notation Q_ℓ we have

$$Q_\ell = \Gamma^{-1}\Big(\rho(\Gamma(P_\ell))\Big). \tag{5.19}$$

At variable nodes, the corresponding density is easier to compute. At a variable node of degree j, the incoming messages from check nodes are summed along with the message coming from the channel. Let P_ℓ^0 denote the density of the message coming from the channel. By the independence assumption, the messages from check nodes are i.i.d. according to Q_ℓ and the channel message is independent from all other incoming messages. Then, the density of the messages emanating from a variable node of degree j towards the check nodes, denoted by $P_{(\ell+1),j}$ is simply the standard convolution over $\mathbb{R}$ of all incoming messages, i.e.,

$$P_{(\ell+1),j} = P_\ell^0 \otimes Q_\ell^{\otimes(j-1)}. \tag{5.20}$$

Summing over all the possible degrees of the variable node we have

$$\begin{aligned} P_{(\ell+1)} &= \sum_{j\geq 2} \lambda_j P_{(\ell+1),j} \\ &= \sum_{j\geq 2} \lambda_j P_\ell^0 \otimes Q_\ell^{\otimes(j-1)} \\ &= P_\ell^0 \otimes \sum_{j\geq 2} \lambda_j Q_\ell^{\otimes(j-1)}, \end{aligned} \tag{5.21}$$

substituting the expression we have derived for Q_ℓ we take

$$P_{(\ell+1)} = P_\ell^0 \otimes \sum_{j\geq 2} \lambda_j \Bigg(\Gamma^{-1} \Bigg(\sum_{i\geq 2} \rho_i \Gamma(P_\ell)^{\star(i-1)} \Bigg) \Bigg)^{\otimes(j-1)}. \qquad \textbf{5.22}$$

By slightly abusing notation again, the final expression for $P_{(\ell+1)}$ is

$$P_{(\ell+1)} = P_\ell^0 \otimes \lambda\Big(\Gamma^{-1}\big(\rho(\Gamma(P_\ell))\big)\Big). \qquad \textbf{5.23}$$

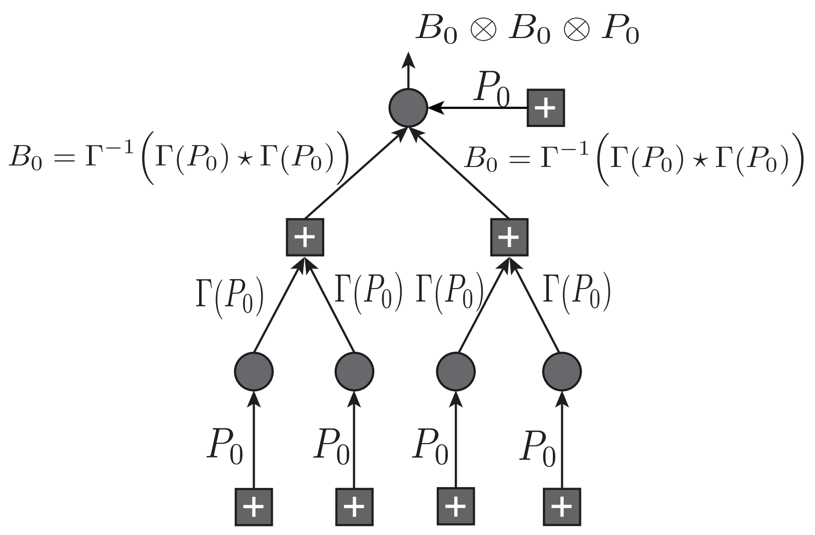
Note that P_ℓ^0 depends only on the channel observation and is actually not a function of ℓ. Moreover, P_ℓ^0 describes the initial L-density for BMS channels assuming that the all-one codeword was transmitted. It can be proved [6] that $P_{(\ell+1)}$ is symmetric. The average probability of error at round $(\ell+1)$ is:

$$P_e(P_{(\ell+1)}) = \int_{-\infty}^{0} P_{(\ell+1)}(x)\ dx. \qquad \textbf{5.24}$$

From corollary 1 in [6], it follows that $P_e(P_{(\ell+1)})$ converges to zero if and only if $P_{(\ell+1)}$ converges to Δ_∞ (= unit mass at infinity). Important sufficient conditions such as monotonicity and stability can be found in [6].

In order to understand better the aforementioned rules for the densities of variable-to-check and check-to-variable messages, we present an example in **Fig. 5.1**.

P_0 denotes the density of $\log \frac{p(y_i/x_i=1)}{p(y_i/x_i=-1)}$ given $X_i = 1$.



$B_0 \otimes B_0 \otimes P_0$
P_0
$B_0 = \Gamma^{-1}\Big(\Gamma(P_0) \star \Gamma(P_0)\Big)$
$B_0 = \Gamma^{-1}\Big(\Gamma(P_0) \star \Gamma(P_0)\Big)$
$\Gamma(P_0)$
$\Gamma(P_0)$
$\Gamma(P_0)$
$\Gamma(P_0)$
P_0
P_0
P_0
P_0

Figure 5.1: Graphical representation of Density Evolution rules.

5.1.3 Discrete Density Evolution

Density Evolution is too complicated for direct use. A computer implementation becomes possible through discrete density evolution, i.e., quantizing the message alphabet and studying the evolution of pmfs instead of pdfs. Here we provide a qualitative description [11] of density evolution and the formulation of discrete density evolution for the Belief-propagation algorithm.

In the first iteration, the decoder is initialized with messages from the channel. This is the initial density of messages from variable nodes to check nodes. Given this density and knowing the update rule at the check nodes, we can compute the density of the output messages from the check nodes. Considering the update rule for a check node with two inputs whose input densities are given as pmf_1 and pmf_2 and assuming a uniform quantization of the messages with a quantization interval Δ, the pmf of the output can be computed as

$$\text{pmf}[k] = \sum_{(i,j):k\cdot\Delta=\mathcal{R}(i\Delta,j\Delta)} \text{pmf}_1[i]\text{pmf}_2[j], \qquad \textbf{5.25}$$

where

$$\mathcal{R}(x,y) = Q\left(2\tanh^{-1}\left(\tanh(\frac{x}{2})\tanh(\frac{y}{2})\right)\right).$$

We denote the probability mass function (pmf) of a quantized message $\overline{w}$ by $p_w[k] = Pr(\overline{w} = k\Delta)$ for $k \in \mathbb{Z}$. Here $Q(\cdot)$ is the quantization function defined as:

$$Q(w) = \begin{cases} \lfloor \frac{w}{\Delta} + \frac{1}{2} \rfloor \cdot \Delta, & \text{if } w \geq \frac{\Delta}{2}, \\ \lceil \frac{w}{\Delta} - \frac{1}{2} \rceil \cdot \Delta, & \text{if } w \leq -\frac{\Delta}{2}, \\ 0, & \text{otherwise.} \end{cases}$$

For check nodes of higher degree, the output density can be computed by noticing that the sum-product check node update rule satisfies

$$\text{CHK}(m_1, m_2, \cdots, m_{d_c-1}) = \text{CHK}(m_1, \text{CHK}(m_2, \cdots, m_{d_c-1})). \qquad \textbf{5.26}$$

Knowing the output density (pmf) of check nodes and the update rule at the variable nodes we can find the message density at the output of the variable nodes. Considering a variable node with two inputs whose pmfs are given by pmf_1 and pmf_2 the output pmf for the update rule can be computed as

$$\text{pmf}[n] = \text{pmf}_1[n] \circledast \text{pmf}_2[n], \qquad \textbf{5.27}$$

where $\circledast$ represents discrete convolution. For variable node of higher degrees we use the fact that

$$\mathrm{VAR}(m_1, m_2, \cdots, m_{d_\mathrm{v}-1}) = \mathrm{VAR}(m_1, \mathrm{VAR}(m_2, \cdots, m_{d_\mathrm{v}-1})). \quad \mathbf{5.28}$$

This finishes density evolution for one iteration of decoding. We can repeat this task for as many iterations as required and find the density of messages at each iteration. Assuming that a "0" information bit is mapped to a +1 signal on the channel and a "1" information bit is mapped to a −1 signal, the message error rate is the negative tail of the density, since a negative message is carrying a belief for a "1" information bit. The negative tail of density should vanish if the decoding is successful. The proofs for the validity and an analysis of accuracy of discrete density evolution can be found in [11].

5.1.4 LDPC Code Design

In this thesis, we present the optimization technique presented by S. Y. Chung in [3, p.197]. We remind that the rate for a given degree distribution pair is

$$r(\lambda, \rho) \triangleq 1 - \frac{\sum_{i \geq 2} \rho_i / i}{\sum_{i \geq 2} \lambda_i / i}.$$

The purpose is to maximize the rate of the code while maintaining some constraints:

1. $\lambda(1) = 1$ and $\lambda_i \geq 0$, $2 \leq i \leq d_\mathrm{v}$;
2. the new $\lambda(x)$ is not significantly different from the old one (required to guarantee that the linear programming formulation is valid);
3. the new $\lambda(x)$ is better than the old one (produces smaller probability of error).

Let $\lambda(x) = \sum_{i=2}^{d_\mathrm{v}} \lambda_i x^{i-1}$ denote the current left degree distribution and and let $\lambda'(x) = \sum_{i=2}^{d_\mathrm{v}} \lambda'_i x^{i-1}$ denote the updated (hopefully improved) left degree distribution.
Initially, we choose $\lambda(x) = x^{d_\mathrm{v}-1}$, to make the initial rate low.
Let e_ℓ denote the probability of error (i.e. probability of being negative) of the input message to check nodes at the ℓ-th iteration when $\lambda(x)$ is used, i.e.,

$$e_\ell = \sum_{i=2}^{d_\mathrm{v}} \lambda_i e_{\ell,i}, \quad \mathbf{5.29}$$

where $e_{\ell,i}$ is the probability of error of the output message of the degree-i variable node at the ℓ-th iteration. Similarly, we define e'_ℓ as the probability of error of the input message to check nodes at the ℓ-th iteration when $\lambda(x)$ is used up to $(\ell-1)$-th iteration and $\lambda'(x)$ is used at the ℓ-th iteration. Note that e'_ℓ is linear in λ_i's, $i = 2, \cdots, d_{\text{v}}$.
We use the following for the constraint 2 above:

$$|e'_\ell - e_\ell| \le \max[0, \delta(e_{(\ell-1)} - e_\ell)],\ 1 \le \ell \le m \qquad \mathbf{5.30}$$

where $\delta \ll 1$ is a small positive number and m is the maximum number of iterations. Note that this constraint is linear in λ'_i's. The "max" operation is not needed if the density evolution for the belief-propagation algorithm is perfect. However, since the probability of error can increase for the discrete density evolution, we need to make sure that the right hand side of the aforementioned inequality is non-negative.
The following is the constraint 3 above:

$$e'_\ell \le e_{(\ell-1)},\ 1 \le \ell \le m. \qquad \mathbf{5.31}$$

Notice that all constraints here and the objective function (rate) are linear in λ'_i's, $2 \le i \le d_{\text{v}}$.
We first fix $\rho(x)$, then choose the initial $\lambda(x)$. We run the above linear program several times until it converges. If the resulting rate is not equal to the desired rate, then we change $\rho(x)$ and we repeat the steps. Sometimes, we need to try several different initial $\lambda(x)$'s for better results. We use concentrated $\rho(x)$'s only, $\rho(x) = (1-\rho)x^{j-1} + \rho x^j$ for some integer $j \ge 2$. This restriction not only makes it easier to optimize $\rho(x)$, especially for large maximum variable degrees, but also is not too restrictive since codes optimized in this way are almost as good as codes with both degree distributions optimized. The average check degree ρ_{av} is used to parametrize $\rho(x)$ where $\rho_{\text{av}} = (1-\rho)(j-1) = j - 1 + \rho$.
Some very good half-rate codes are illustrated in Table 8.4 in [3]. For instance, a very good degree distribution pair with $\rho_{\text{av}} = 17$,which is 0.00495dB of the Shannon limit for $n = 1e7$, is:

$$\lambda(x) = 0.105332x + 0.103764x^2 + 0.035237x^5 + 0.100941x^6 + 0.029112x^{14}$$
$$+0.099561x^{19} + 0.084268x^{49} + 0.034889x^{69} + 0.088540x^{149} + 0.027687x^{249}$$
$$+0.059494x^{399} + 0.097641x^{899} + 0.070928^{2999} + 0.062606x^{3999}$$

since $j - 1 + \rho = 17 \rightarrow \rho(x) = x^{17}$.

For the following simulation we used 1200 decoding iterations and block-length 10^6.

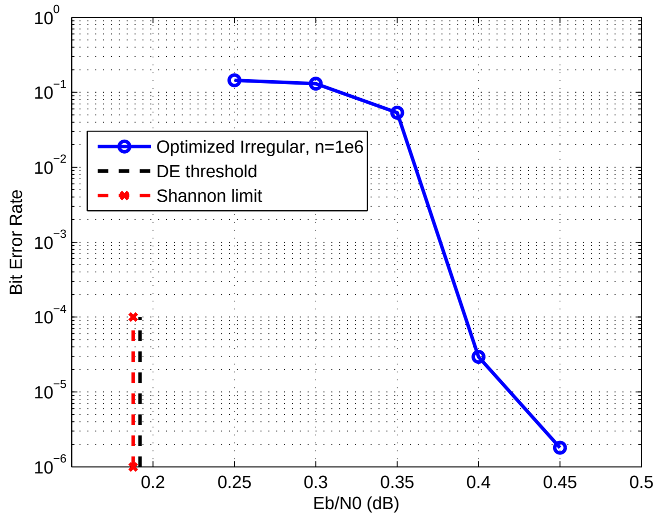


Figure 5.2: Bit Error Rate of an optimized irregular LDPC code.

5.2 Gaussian Approximation

In [5], it has been shown that starting with a *symmetric* pdf, i.e., a pdf $f(x)$ that satisfies $f(x) = e^x f(-x)$, and using belief-propagation decoding, the pdf of LLR messages in the decoder remains *symmetric*.

A Gaussian pdf with mean μ and variance σ^2 is *symmetric* if, and only if, $\sigma^2 = 2\mu$ [12]. As a result, a *symmetric Gaussian* density can be expressed by a *single* parameter. Interestingly, the density of LLR intrinsic messages for a Gaussian channel is symmetric, and hence, under belief-propagation decoding, remains symmetric.

As stated earlier, Density Evolution is an algorithm for computing the capacity of LDPC codes under message-passing decoding. For memoryless binary-input continuous-output additive white Gaussian noise channels and sum-product decoders, the authors in [12] use a Gaussian approximation for message densities under Density Evolution to simplify the analysis of the decoding algorithm. Therefore, the infinite-dimensional problem of iteratively tracking message densities, which is needed to find the exact threshold, is reduced to a 1-D problem of updating *means* of *symmetric Gaussian* densities. This simplification not only allows to calculate the threshold quickly and to understand the behaviour of the decoder better, but also makes it easier to design good irregular LDPC codes for AWGN channels. Finally, Gaussian approximation is based on approximating message densities as Gaussians for regular LDPC codes and as Gaussian mixtures for irregular LDPC codes. We remind that the prerequisites for this analysis are:

1. The Tanner graph of the LDPC codes is cycle-free;
2. The all-one codeword is transmitted;
3. During each iteration, due to 1, the messages are independent and identically distributed and, more specifically, they follow the *symmetric Gaussian* distribution.

5.2.1 Gaussian Approximation for Regular LDPC codes

At first, we assume regular (d_{v}, d_c) LDPC codes and remind the BP rules.
At variable nodes:

$$\mathrm{v} = \sum_{i=0}^{d_{\mathrm{v}}-1} u_i, \qquad \textbf{5.32}$$

where v is a message in LLR form from a variable node to a check node and $\{u_i\}_{i=0}^{d_{\mathrm{v}}-1}$ are the incoming LLRs from the neighbours of the variable node except the check node that gets the message v. u_0 is the observed (channel) LLR of the output bit associated with the variable node. The LLR message u_0 from the channel is Gaussian with mean $\frac{2}{\sigma_n^2}$ and variance $\frac{4}{\sigma_n^2}$, where σ_n^2 is the variance of the channel noise. Thus, if all $\{u_i,\, i \geq 1\}$ [which are i.i.d] are Gaussian in **5.32**, then the resulting sum is also Gaussian because it is the sum of independent Gaussian random variables.
At check nodes:

$$\tanh\left(\frac{u}{2}\right) = \prod_{j=1}^{d_c-1} \tanh\left(\frac{\mathrm{v}_j}{2}\right), \qquad \textbf{5.33}$$

where $\{\mathrm{v}_j\}_{j=1}^{d_c-1}$ are the incoming LLRs from $d_c - 1$ neighbours of a check node, and u is the message sent to the remaining neighbour.
Assuming *symmetric Gaussian* distributions, that is, $\mathrm{v} \sim \mathcal{N}(\mu_{\mathrm{v}}, 2\mu_{\mathrm{v}})$ and $u \sim \mathcal{N}(m_u, 2\mu_u)$, and independent messages, during the ℓ-th BP round (variable-to-check and, back, check-to-variable) so for a degree-d_{v} variable node we obtain

$$\mu_{\mathrm{v}}^{(\ell)} = \mu_{u_0} + (d_{\mathrm{v}} - 1)\mu_u^{(\ell-1)}, \qquad \textbf{5.34}$$

where μ_{u_0} is the mean of u_0 and ℓ denotes the ℓ-th round. We omit the index i because, as explained earlier, the u_i's are i.i.d. for $1 \le i \le d_{\mathrm{v}}$ and have the same mean μ_u. Note that $\mu^{(0)} = 0$ since the initial message from any check node is 0.
The update mean $\mu_u^{(\ell)}$ at the ℓ-th round can be calculated by taking expectations on each side of **5.33**, i.e.,

$$\mathcal{E}\left[\tanh\left(\frac{u^{(\ell)}}{2}\right)\right] = \left\{\mathcal{E}\left[\tanh\left(\frac{\mathrm{v}^{(\ell)}}{2}\right)\right]\right\}^{d_c-1}, \qquad \mathbf{5.35}$$

where we have omitted the index j and simplified the product because the v_j's are i.i.d. One complete round begins at variable nodes and then ends at the check nodes. Note that the expectation $\mathcal{E}\left[\tanh\frac{u}{2}\right]$ depends only on the mean μ_u of u, since u is Gaussian with mean μ_u and variance $2\mu_u$; i.e.,

$$\mathcal{E}\left[\tanh\frac{u}{2}\right] = \frac{1}{\sqrt{4\pi\mu_u}}\int_{\mathbb{R}} \tanh\left(\frac{u}{2}\right)\cdot e^{-\frac{(u-\mu_u)^2}{4\mu_u}}\,du. \qquad \mathbf{5.36}$$

We define the following function $\phi(x)$ for $x \in [0,+\infty)$, which will be useful and convenient for further analysis.

Definition 5.2. ([12], Definition 1).

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}}\int_{\mathbb{R}} \tanh\left(\frac{u}{2}\right)\cdot e^{-\frac{(u-x)^2}{4x}}\,du, & \text{if } x > 0; \\ 1, & \text{if } x = 0. \end{cases} \qquad \mathbf{5.37}$$

It is easy to check that $\phi(x)$ is continuous and monotonically decreasing on $[0,+\infty)$, with $\phi(0) = 1$ and $\phi(\infty) = 0$. ◇

For practical purposes the following approximation for $\phi(x)$ is commonly used

$$\hat{\phi}(x) = \begin{cases} e^{-0.4527x^{0.86}+0.0218}, & \text{if } x < 10; \\ \sqrt{\frac{\pi}{x}}\left(1-\frac{20}{7x}\right), & \text{if } x \ge 10. \end{cases}$$

The first part of $\hat{\phi}(x)$ $(x < 10)$ is easily invertible. The second one, is a fairly well-behaved function and can be inverted using a lookup table.
Is is easy to see that, if $u \sim \mathcal{N}(\mu_u, 2\mu_u)$, then

$$\mathcal{E}\left[\tanh\frac{u}{2}\right] = 1 - \phi(\mu_u). \qquad \mathbf{5.38}$$

Substituting the aforementioned equation in **5.35** we have

$$\begin{aligned}1-\phi(\mu_u^{(\ell)}) &= \left[1-\phi(\mu_{\mathrm{v}}^{(\ell)})\right]^{d_c-1}\\ &\overset{5.34}{=} \left[1-\phi(\mu_{u_0}+(d_{\mathrm{v}}-1)\mu_u^{(\ell-1)})\right]^{d_c-1}.\end{aligned} \qquad \mathbf{5.39}$$

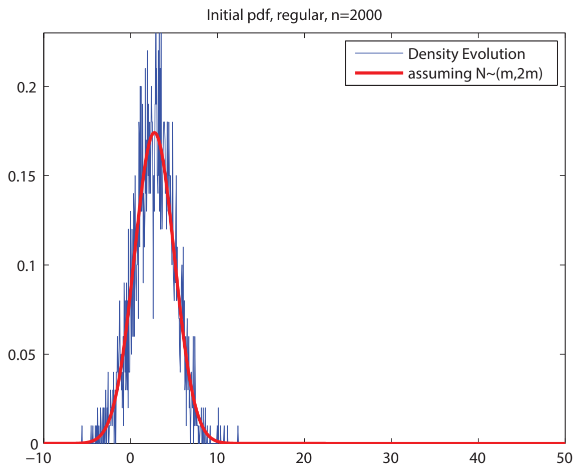
Thus, the update rule for μ_u^ℓ becomes

$$\mu_u^\ell = \phi^{-1}\left(1-\left[1-\phi(\mu_{u_0}+(d_{\mathrm{v}}-1)\mu_u^{(\ell-1)})\right]^{d_c-1}\right) \qquad \mathbf{5.40}$$

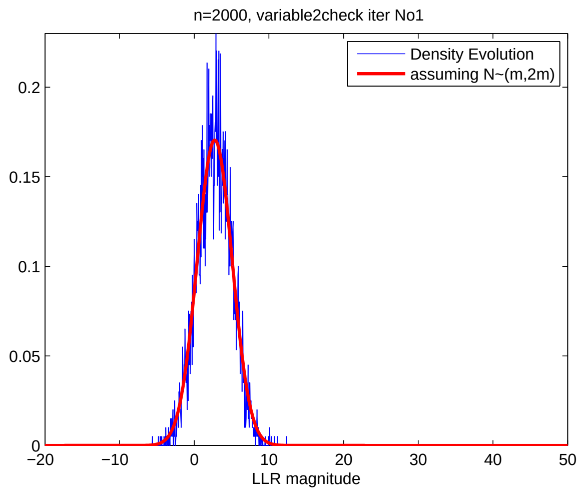
with $\mu_u^{(0)} = 0$.

We will now illustrate Density Evolution together with Gaussian approximation for the $(3,6)$ LDPC ensemble over the BI-AWGN channel. The threshold for this ensemble has been found to be $\sigma^* = 0.8747$. We will present the message densities after a certain number of iterations and we will set firstly the noise variance equal to $\sigma = 0.8511 < \sigma^*$ and secondly $\sigma = 0.9411 > \sigma^*$.

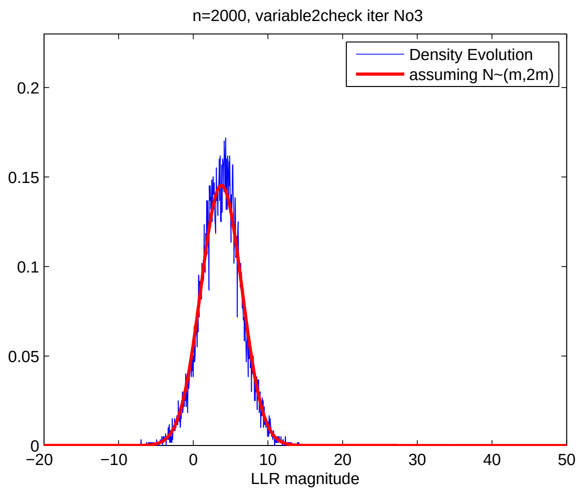
Variable-to-check message densities:



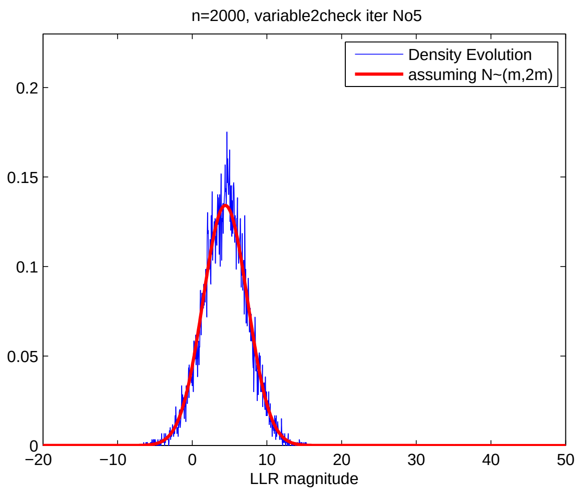
(a) Initial LLR distribution



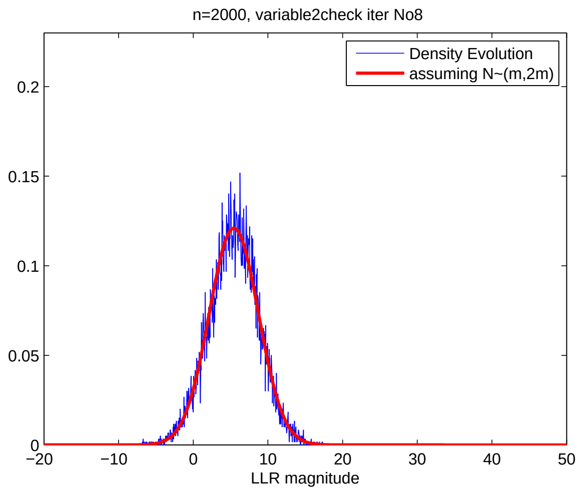
(b) Iteration 1



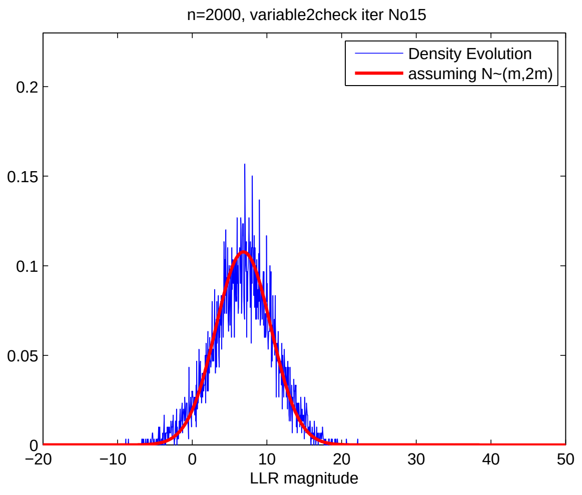
(c) Iteration 3



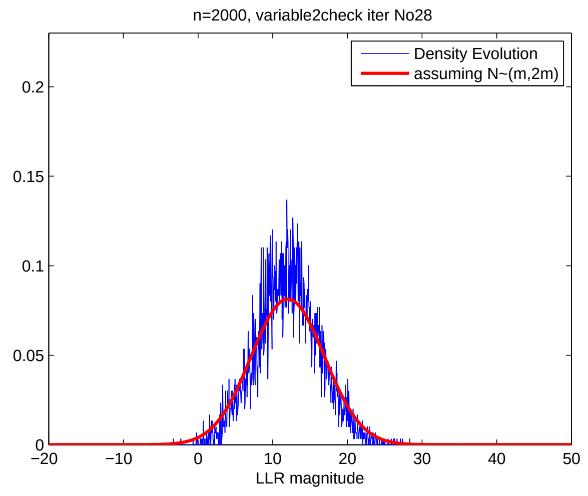
(d) Iteration 5



(e) Iteration 8



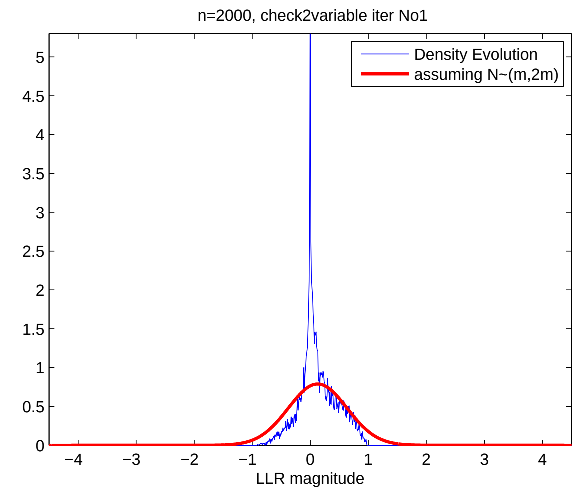
(f) Iteration 15



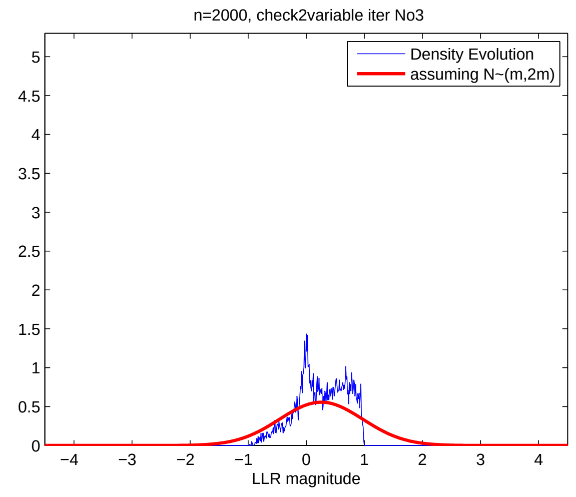
(g) Iteration 28

Figure 5.3: Density Evolution and Gaussian approximation for the BI-AWGN channel for $\sigma < \sigma^*$.

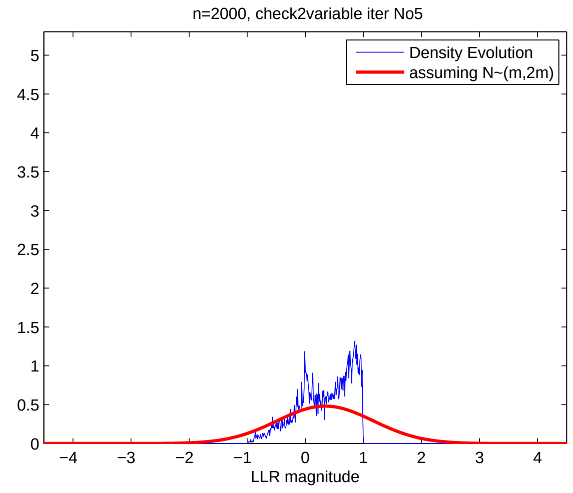
Check-to-variable messages densities:



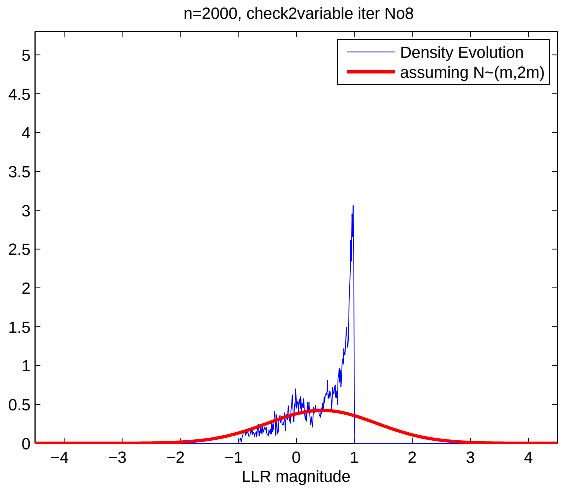
(a) Iteration 1



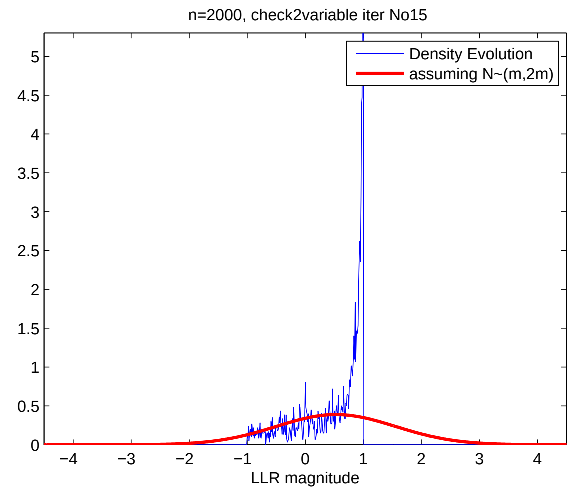
(b) Iteration 3



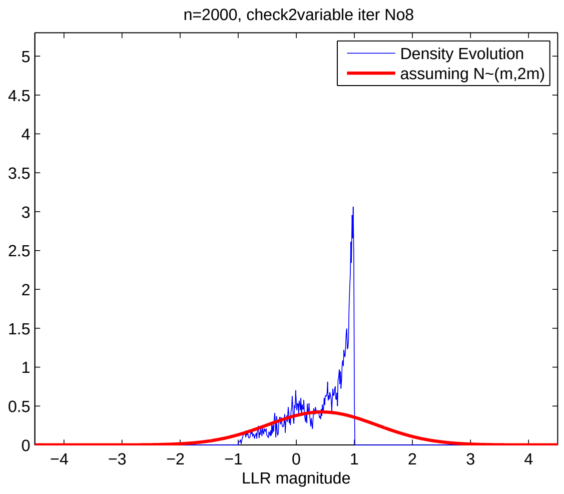
(c) Iteration 5



(d) Iteration 8



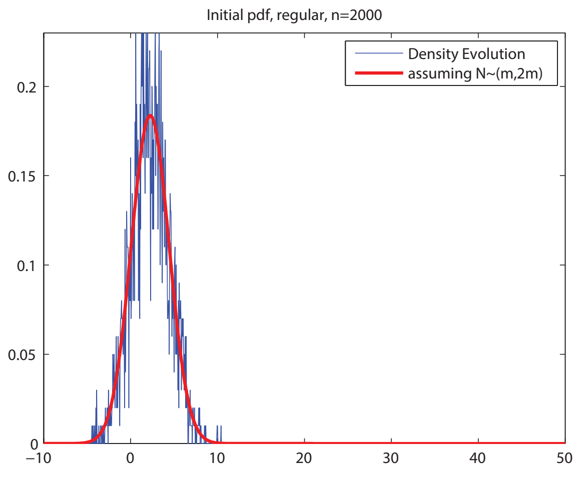
(e) Iteration 15



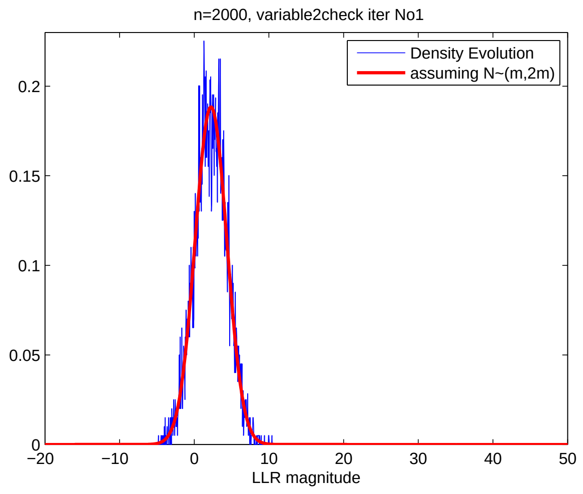
(f) Iteration 28

Now will present the message densities for $\sigma > \sigma^*$. We will observe that the densities start moving to the right, but return to a fixed point after a certain number of iterations, leading to a non-zero probability of error. In this case, even after infinite number of iterations, the message density will not tend to a point mass at $+\infty$.

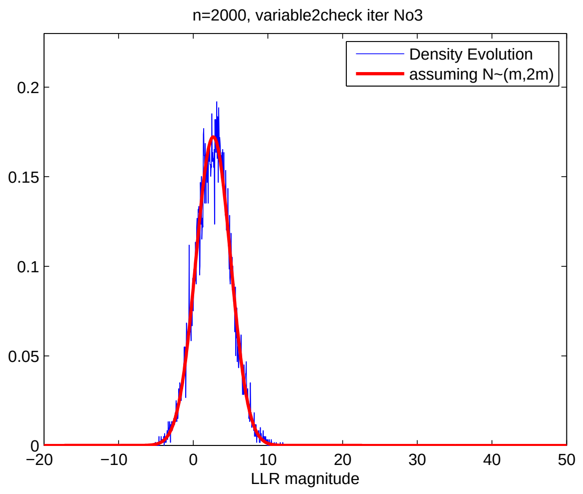
Variable-to-check message densities:



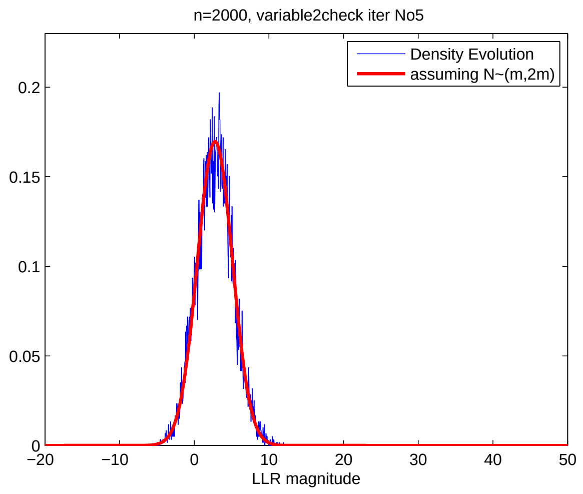
(g) Initial LLR distribution



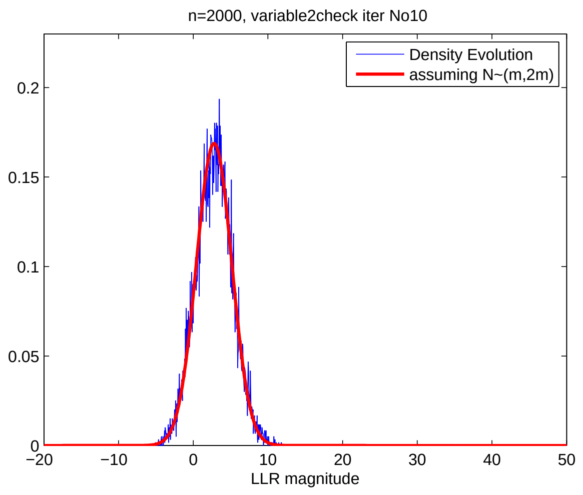
(h) Iteration 1



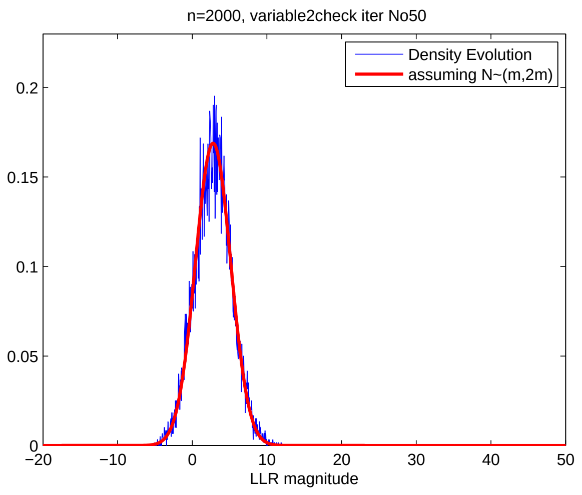
(i) Iteration 3



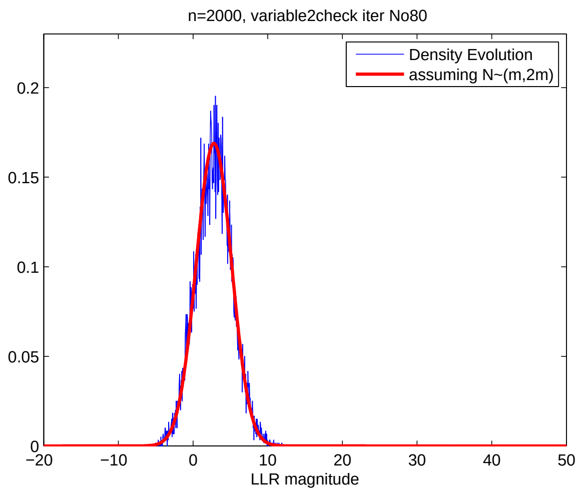
(j) Iteration 5



(k) Iteration 10



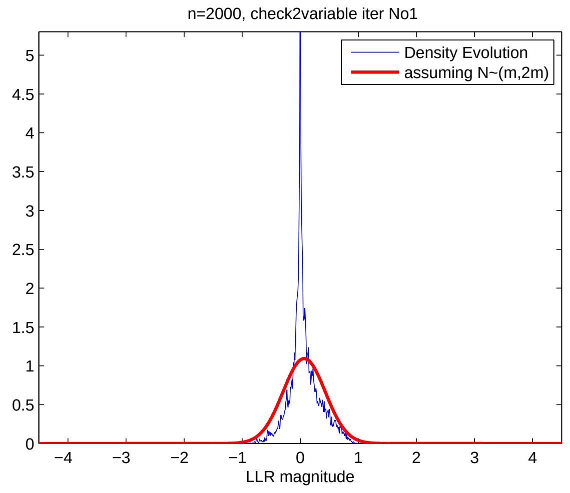
(l) Iteration 50



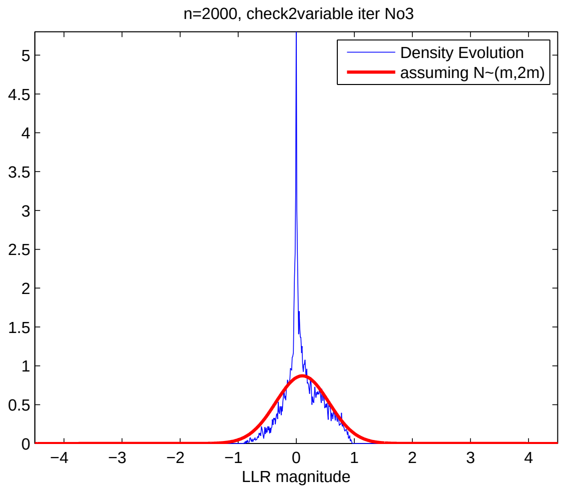
(m) Iteration 80

Figure 5.4: Density Evolution and Gaussian approximation for the BI-AWGN channel for $\sigma > \sigma^*$.

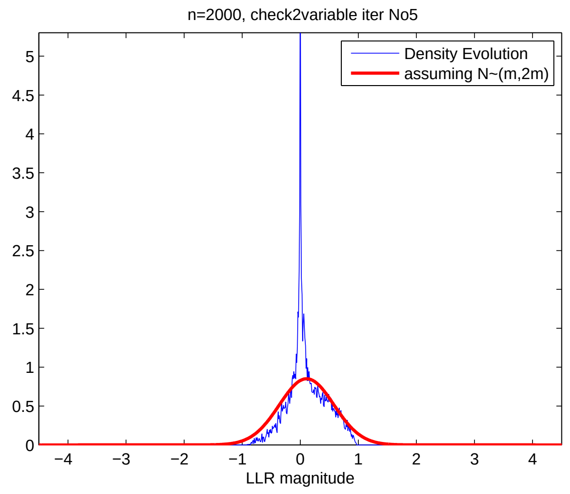
Check-to-variable messages densities:



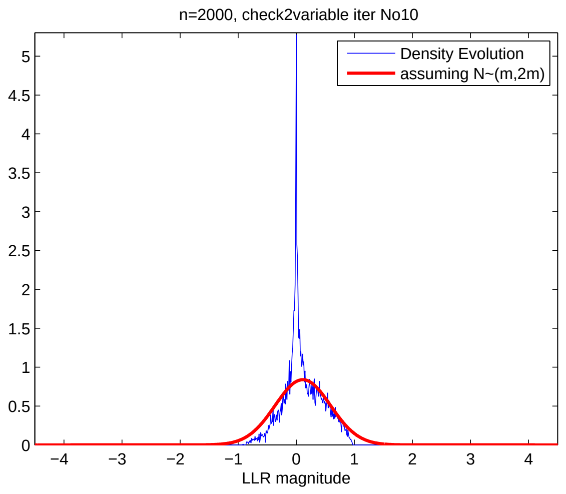
(a) Iteration 1



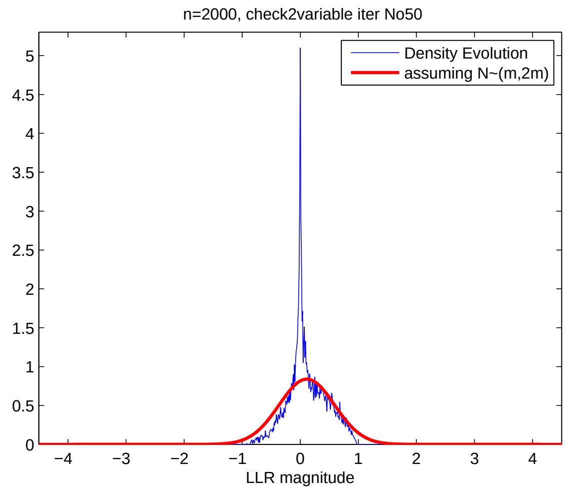
(b) Iteration 3



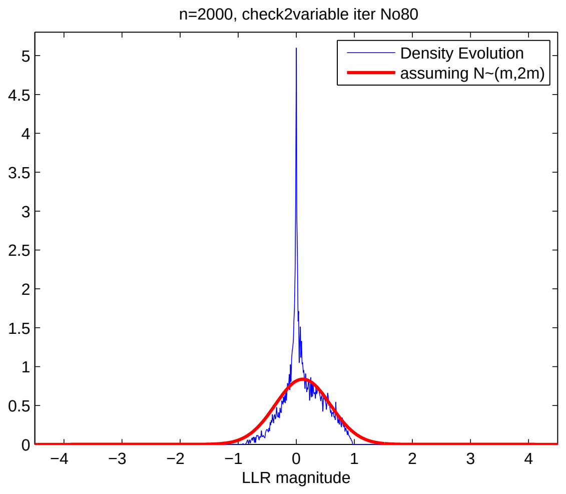
(c) Iteration 5



(d) Iteration 10



(e) Iteration 50



(f) Iteration 80

5.2.2 Gaussian Approximation for Irregular LDPC codes

We again consider an ensemble of random codes with degree distributions $\lambda(x) = \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i x^{i-1}$ and $\rho(x) = x^{d_c-1}$. We assume that $u_i^{(\ell)}$'s are i.i.d. symmetric Gaussian. Then, the message leaving a variable node of degree i is symmetric Gaussian with mean

$$\mu_{\mathrm{v},i}^{(\ell)} = \mu_{u_0} + (i-1)\mu_u^{(\ell-1)}, \qquad \mathbf{5.41}$$

where μ_{u_0} is the mean u_0 and $\mu_u^{(\ell-1)}$ is the mean of u at the $(\ell-1)$-th iteration, where u is a Gaussian mixture in general (when $\rho(x)$ is not concentrated in one degree [3, p.168]). The variance of the output density is given by $2\mu_{\mathrm{v},i}^{(\ell)}$. Therefore, at the ℓ-th iteration, using the total probability theorem an incoming message v to a check node will have the following Gaussian mixture density:

$$f_{\mathrm{v}^{(\ell)}}(x) = \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \mathcal{N}\Big(x, \mu_{\mathrm{v},i}^{(\ell)}, 2\mu_{\mathrm{v},i}^{(\ell)}\Big), \qquad \mathbf{5.42}$$

where $\mu_{\mathrm{v},i}^{(\ell)}$ is the mean of the Gaussian output from a degree-i variable node. Using the previous equation we get:

$$\begin{aligned}
\mathcal{E}\Big\{\tanh\frac{\mathrm{v}^{(\ell)}}{2}\Big\} &= \int_{\mathbb{R}} \tanh\Big(\frac{x}{2}\Big) f_{\mathrm{v}^{(\ell)}}(x)\, dx \\
&= \int_{\mathbb{R}} \tanh\Big(\frac{x}{2}\Big)\Bigg(\sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \mathcal{N}\Big(x, \mu_{\mathrm{v},i}^{(\ell)}, 2\mu_{\mathrm{v},i}^{(\ell)}\Big)\Bigg)\, dx \\
&= \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \int_{\mathbb{R}} \tanh\Big(\frac{x}{2}\Big)\mathcal{N}\Big(x, \mu_{\mathrm{v},i}^{(\ell)}, 2\mu_{\mathrm{v},i}^{(\ell)}\Big)\, dx \\
&= \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \Big(1 - \phi(\mu_{\mathrm{v},i}^{(\ell)})\Big) \\
&= 1 - \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi(\mu_{\mathrm{v},i}^{(\ell)}). \qquad \mathbf{5.43}
\end{aligned}$$

Thus, for the message $u^{(\ell)}$ that leaves a check node of degree j, we have

$$\begin{aligned}
\mathcal{E}\Bigg[\tanh\Big(\frac{u_j^{(\ell)}}{2}\Big)\Bigg] &= \Bigg\{\mathcal{E}\Bigg[\tanh\Big(\frac{\mathrm{v}^{(\ell)}}{2}\Big)\Bigg]\Bigg\}^{j-1} \\
\Rightarrow \Big(1 - \phi(\mu_{u_j}^{(\ell)})\Big) &= \Bigg(1 - \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi(\mu_{\mathrm{v},i}^{(\ell)})\Bigg)^{j-1} \\
\Rightarrow \mu_{u_j}^{(\ell)} &= \phi^{-1}\Bigg(1 - \Bigg(1 - \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi(\mu_{\mathrm{v},i}^{(\ell)})\Bigg)^{j-1}\Bigg), \qquad \mathbf{5.44}
\end{aligned}$$

and the mean of $u^{(\ell)}$ is given by

$$\begin{aligned}\mu_u^{(\ell)} &= \sum_{j=2}^{d_c} \rho_j \mu_{u_j}^{(\ell)} \\ &= \sum_{j=2}^{d_c} \rho_j \phi^{-1}\left(1 - \left(1 - \sum_{i=2}^{d_\mathrm{v}} \lambda_i \phi(\mu_{\mathrm{v},i}^{(\ell)})\right)^{j-1}\right) \\ &= \sum_{j=2}^{d_c} \rho_j \phi^{-1}\left(1 - \left(1 - \sum_{i=2}^{d_\mathrm{v}} \lambda_i \phi(\mu_{u_0} + (i-1)\mu_u^{(\ell-1)})\right)^{j-1}\right). \qquad \mathbf{5.45}\end{aligned}$$

Unfortunately, the aforementioned relation is *not* useful for code design because it is *not* linear in the λ_i. However, we can derive an analogous expression [12] linear in the λ_i. We define for $0 < s < \infty$, $0 < r \leq 1$ and concentrated $\rho(x)$,

$$\begin{aligned}h_i(s,r) &= \phi\Big(s + (i-1)\phi^{-1}(1-(1-r)^{d_c-1})\Big); \\ h(s,r) &= \sum_{i=2}^{d_\mathrm{v}} \lambda_i h_i(s,r). \qquad \mathbf{5.46}\end{aligned}$$

Let us prove it. Let us assume concentrated $\rho(x) = x^{d_c-1}$ and $\mathrm{v}^{(\ell-1)} \sim \mathcal{N}(\mu_\mathrm{v}^{(\ell-1)}, 2\mu_\mathrm{v}^{(\ell-1)})$. Then, at check nodes, we have messages with mean $\mu_u^{(\ell)}$ that can be computed as follows:

$$\begin{aligned}1 - \phi(\mu_u^{(\ell)}) &= \left(1 - \phi(\mu_\mathrm{v}^{(\ell-1)})\right)^{d_c-1} \\ \mu_u^{(\ell)} &= \phi^{-1}\left(1 - \left(1 - \phi(\mu_\mathrm{v}^{(\ell-1)})\right)^{d_c-1}\right). \qquad \mathbf{5.47}\end{aligned}$$

Then, for a degree-i variable node, we obtain that $\mathrm{v}_i^{(\ell)}$ is symmetric Gaussian with mean

$$\mu_{\mathrm{v},i}^{(\ell)} = \mu_u^{(0)} + (i-1)\mu_u^{(\ell-1)}. \qquad \mathbf{5.48}$$

Thus, $\mathrm{v}^{(\ell)}$ is Gaussian mixture of the form

$$\sum_{i\geq 2} \lambda_i \mathcal{N}(\mu_{\mathrm{v},i}^{(\ell)}, 2\mu_{\mathrm{v},i}^{(\ell)}).$$

From **5.43**, we have

$$\begin{aligned}\mathcal{E}\left\{\tanh\frac{\mathrm{v}^{(\ell)}}{2}\right\} &= 1 - \sum_{i=2}^{d_\mathrm{v}} \lambda_i \phi(\mu_{\mathrm{v},i}^{(\ell)}) \\ &= 1 - \sum_{i=2}^{d_\mathrm{v}} \lambda_i \phi(\mu_u^{(0)} + (i-1)\mu_u^{(\ell)}) \\ &\overset{5.47}{=} 1 - \sum_{i=2}^{d_\mathrm{v}} \lambda_i \phi\left(\mu_u^{(0)} + (i-1)\phi^{-1}\left(1 - \left(1 - \phi(\mu_\mathrm{v}^{(\ell-1)})\right)^{d_c-1}\right)\right). \qquad \mathbf{5.49}\end{aligned}$$

In addition, the aforementioned relation can be written as

$$1 - \phi(\mu_{\mathrm{v}}^{(\ell)}) = 1 - \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi\left(\mu_u^{(0)} + (i-1)\phi^{-1}\left(1 - \left(1 - \phi(\mu_{\mathrm{v}}^{(\ell-1)})\right)^{d_c - 1}\right)\right)$$
$$\leftrightarrow \phi(\mu_{\mathrm{v}}^{(\ell)}) = \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi\left(\mu_u^{(0)} + (i-1)\phi^{-1}\left(1 - \left(1 - \phi(\mu_{\mathrm{v}}^{(\ell-1)})\right)^{d_c - 1}\right)\right). \qquad \mathbf{5.50}$$

Hence, if we define $r_\ell \triangleq \phi(\mu_{\mathrm{v}}^{(\ell)})$, we obtain

$$r_\ell = \sum_{i=2}^{d_{\mathrm{v}}} \lambda_i \phi\left(\mu_u^{(0)} + (i-1)\phi^{-1}\left(1 - \left(1 - r_{\ell-1}\right)^{d_c - 1}\right)\right). \qquad \mathbf{5.51}$$

Then,

$$r_\ell = h(s, r_{\ell-1}), \qquad \mathbf{5.52}$$

with $s = \mu_u^{(0)}$. The initial value of r_0 is $\phi(s)$. ■

The convergence condition is satisfied if $r > h(s,r)$, $\forall r \in (0, \phi(s))$ [12]. We can design an LDPC code by solving the following *continuous* optimization problem.

$$\boxed{\begin{array}{ll} \max & \sum_{i=2}^{\mathrm{v}_{\max}} \frac{\lambda_i}{i} \\ \text{subject to} & r > h(s,r),\ \forall \in (0, \phi(s)) \\ & \sum_{i=2}^{\mathrm{v}_{\max}} \lambda_i = 1,\ \lambda_i \geq 0,\ \text{for } i = 2, \cdots, \mathrm{v}_{\max}. \end{array}} \qquad \mathbf{5.53}$$

Approximate solutions can be obtained by discretization of parameter $r \in [0, \phi(s))$.

Definition 5.3. **([3, p.169]).** The threshold s^* is the infimum of all s in $\mathbb{R}^+$ such that $r_\ell(s)$ converges to ∞ as $\ell \to \infty$. ◇

Therefore, the threshold in terms of noise variance is equal to $\frac{2}{s^*}$. Since $\phi(x)$ is monotonically decreasing on $0 \leq x < \infty$, we conclude that $h(s,r)$ is monotonically increasing on both $0 < s < \infty$ and $0 \leq r < 1$. By induction, we conclude that for all $s > s^*$, $r_\ell(s) > r_\ell(s^*)$ and $r_\ell(s)$ will converge to ∞.

In the case where $\rho(x) = \sum_{j=2}^{d_c} \rho_j x^{j-1}$ it can be proved [12] that

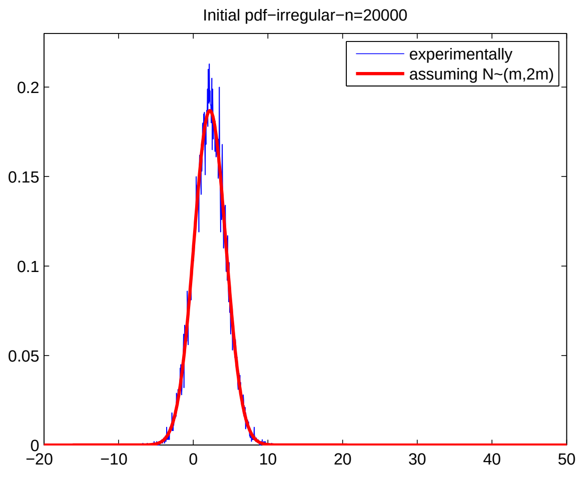
$$r_\ell = \sum_{i=2}^{d_v} \lambda_i \phi\left(\mu_u^{(0)} + (i-1)\sum_{j=2}^{d_c} \rho_j \phi^{-1}\left(1 - \left(1 - r_{\ell-1}\right)^{d_c - 1}\right)\right). \qquad \mathbf{5.54}$$

As for the regular case, we illustrate Density Evolution together with Gaussian Approximation for an irregular LDPC ensemble with degree distributions

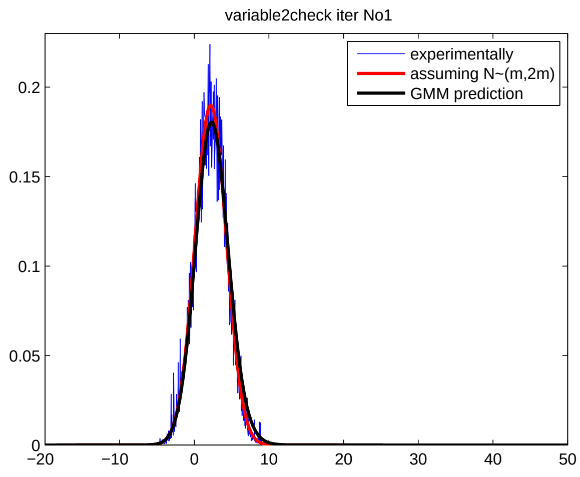
$$\begin{aligned}\lambda(x) &= 0.153425x + 0.147526x^2 + 0.041539x^5 + 0.147551x^6 + 0.047938x^{17} + 0.119555x^{18} \\ &\quad + 0.036379x^{54} + 0.126714x^{55} + 0.179373x^{199}. \\ \rho(x) &= 0.5x^{11} + 0.5x^{13}.\end{aligned}$$

For this ensemble, the threshold in terms of noise variance, is $\sigma^* = 0.97704$. In our simulation, we used $\sigma = 0.9441 < \sigma^*$. We omit the case for $\sigma > \sigma^*$ since as in the regular case, we will observe that the densities start moving to the right, but return to a fixed point after a certain number of iterations, leading to a non-zero probability of error.

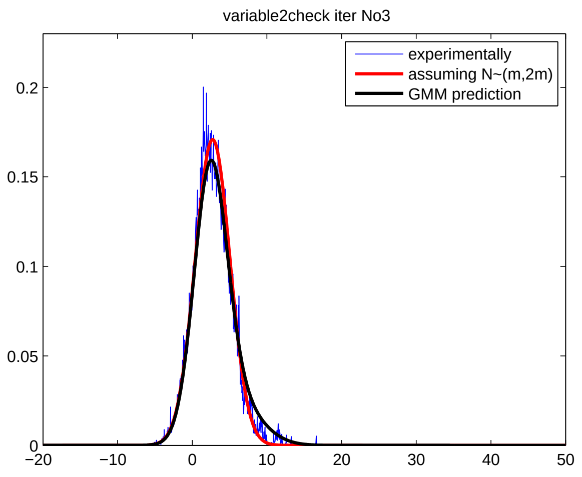
Variable-to-check message densities:



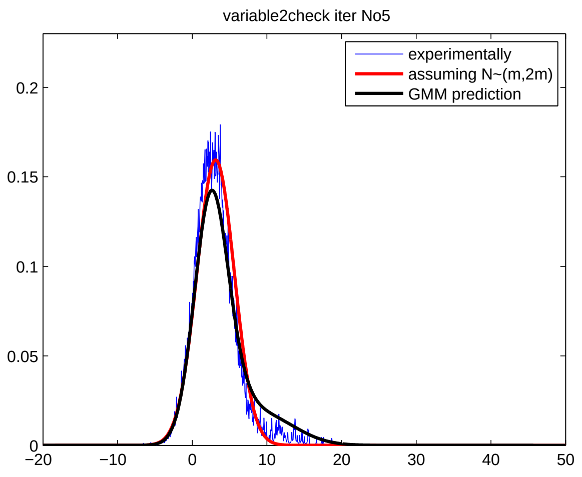
(g) Initial LLR distribution



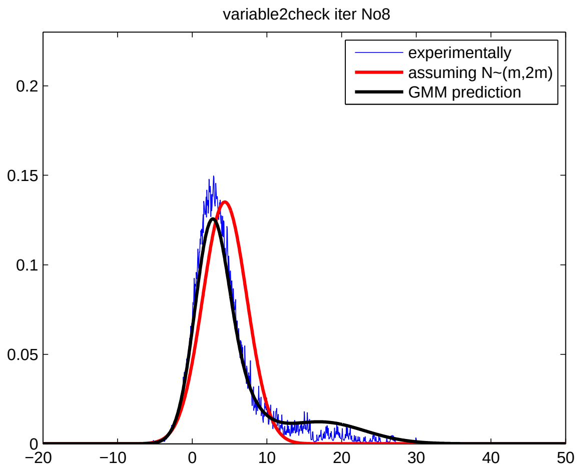
(h) Iteration 1



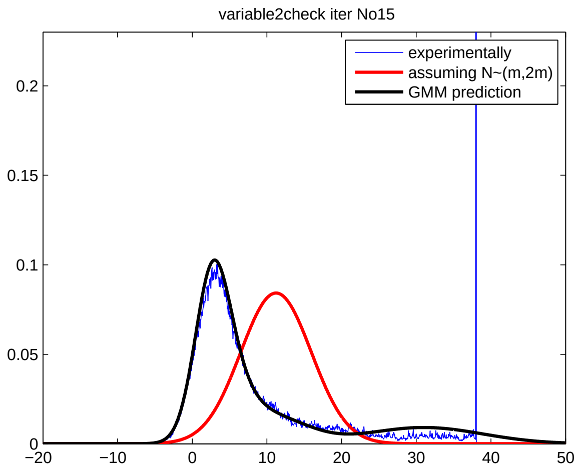
(i) Iteration 3



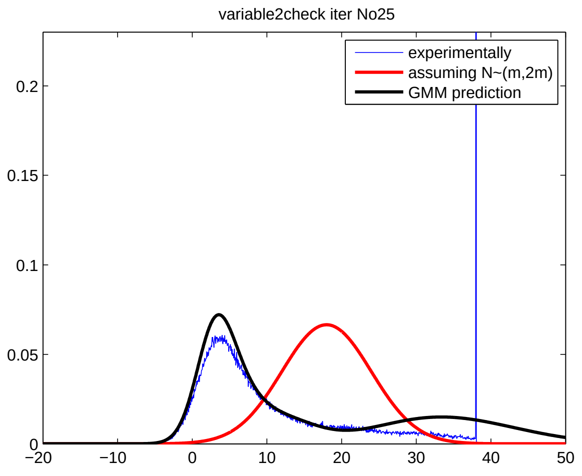
(j) Iteration 5



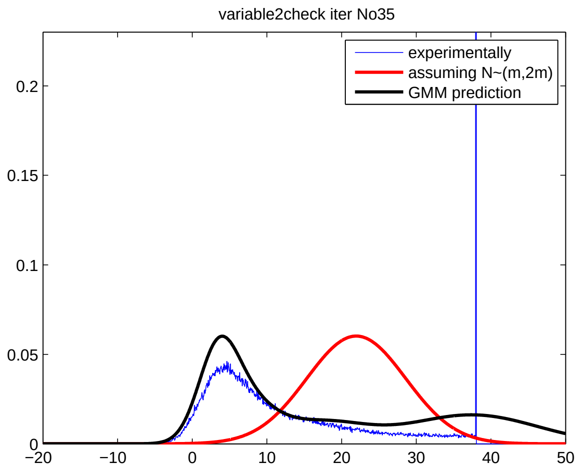
(k) Iteration 8



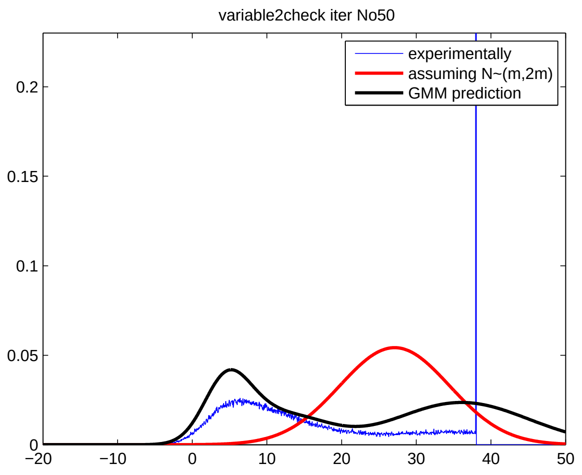
(l) Iteration 15



(m) Iteration 25



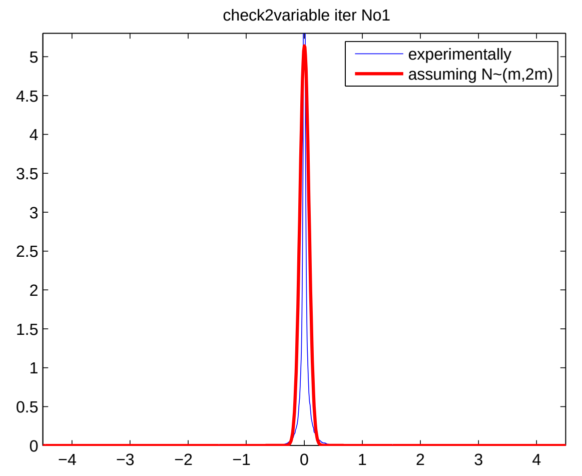
(n) Iteration 35



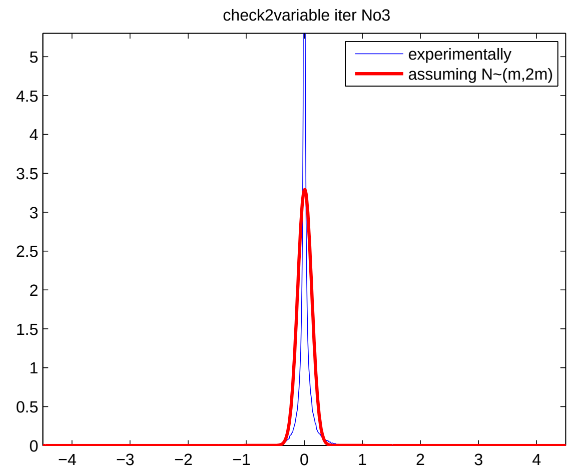
(o) Iteration 50

Figure 5.5: Density Evolution and Gaussian approximation for the BI-AWGN channel for irregular codes with $\sigma < \sigma^*$.

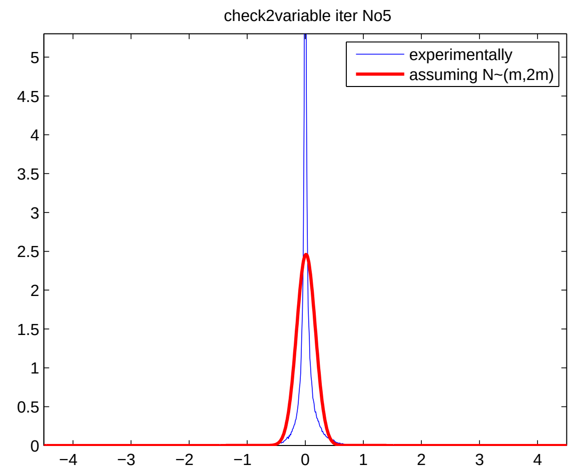
Check-to-variable message densities:



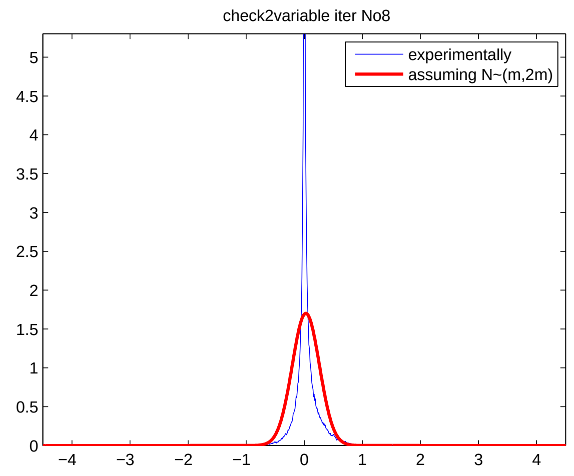
(a) Iteration 1



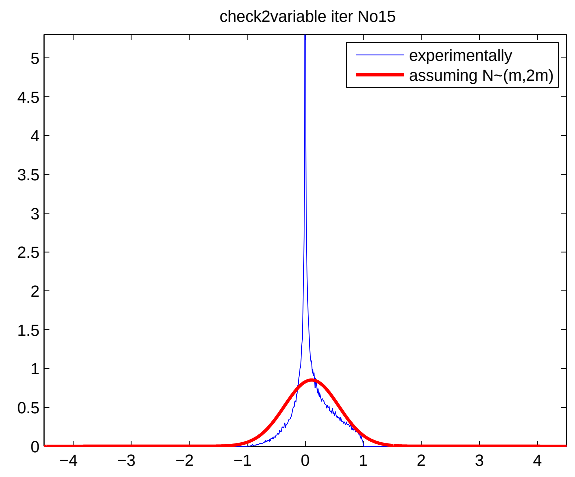
(b) Iteration 3



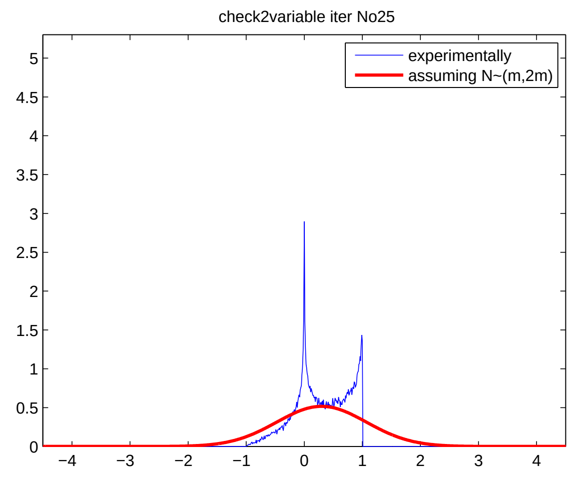
(c) Iteration 5



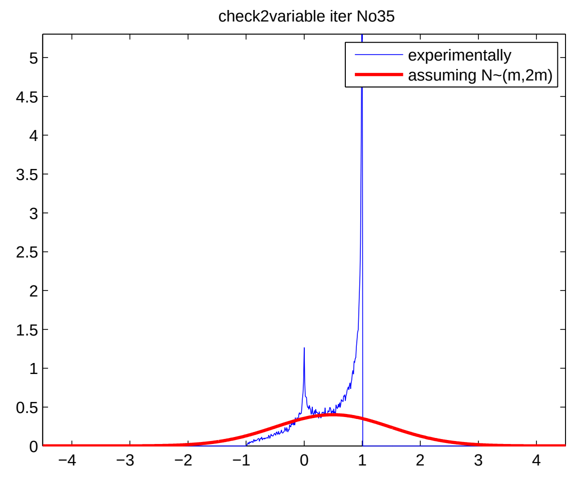
(d) Iteration 8



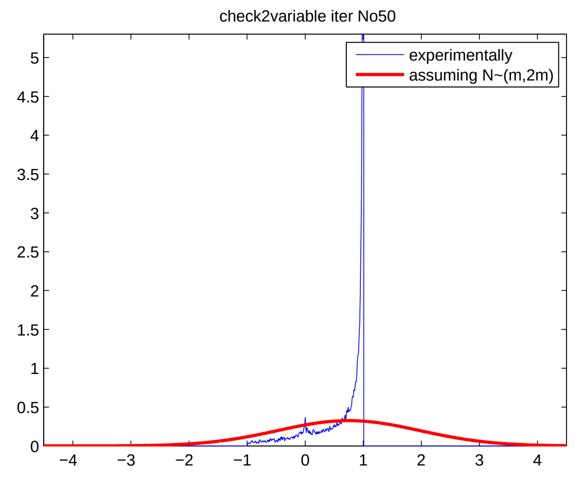
(e) Iteration 15



(f) Iteration 25



(g) Iteration 35



(h) Iteration 50

From the previous figures for irregular codes we can observe that Gaussian approximation is not accurate. This seems to suggest that, as the maximum variable degree of the code increases, the Gaussian approximation becomes less accurate because the irregularity of the code increases, even though the fraction of high-degree variable nodes increases. Hence, we restrict the maximum variable degree to 10 in order to restrict the Gaussian mixture density. Then, it can be inferred that restricting the variable node degree not greater than 10, we limit the range of design rate of the codes between $[0.5, 0.9]$ [12]. For small variable degrees the Gaussian approximation works reasonably well and provides thresholds very close to the corresponding thresholds provided from Density Evolution.

Chapter 6

Relay Channel

6.1 Introduction

The relay channel was first introduced in 1971 by van der Meulen [18] and further studied in 1979 by Cover and El Gamal [19]. The classic work of Cover and El Gamal describes two basic strategies for the relay channel: a Decode-and-Dorward (DF) strategy in which the relay completely decodes the transmitted message and partially forwards the decoded message using a binning technique to allow the complete resolution of the message at the destination, and a more complex compress-and-forward strategy in which the relay does not need to decode the source message. The capacity of the general relay channel is still not known, however, Decode-and-Forward outperforms any other scheme proposed so far when the source-relay channel is strong.

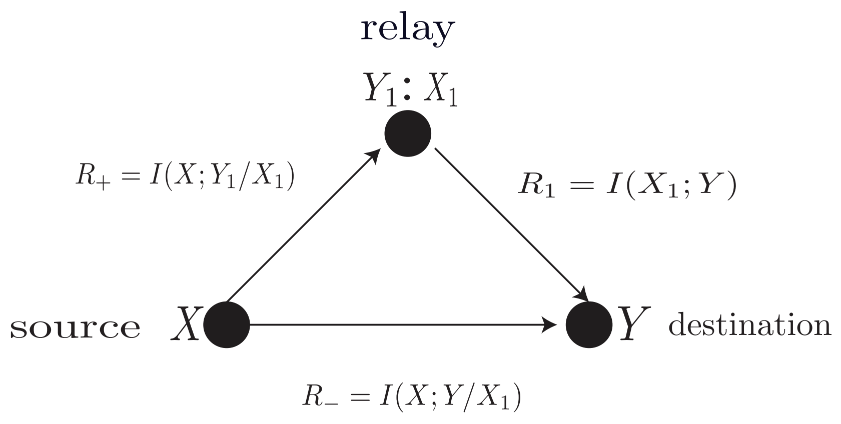


Figure 6.1: Relay Channel from an information theoretic view.

In the sequel, we focus on practical implementation of the DF strategy for half-duplex relay channel. We restrict our attention to Gaussian relay channels at low signal-to-noise ratios (SNRs) for which binary linear block codes are suitable. We show that, within a linear coding framework, the binning strategy in which a bin index of the codeword is transmitted by the relay to the destination can be interpreted as a *parity-forwarding* scheme [7]. Conventional LDPC codes are not suitable since we need to design an LDPC code working at two different channel SNRs: a higher SNR at the relay and a lower SNR at the destination. This represents a novel embedded LDPC code construction named *Bilayer LDPC codes*. In addition, two new ensembles of LDPC codes, *bilayer-expurgated* codes and *bilayer-lengthened* LDPC codes, are proposed to simultaneously approach the capacities of two Gaussian channels at two different SNRs.

The code design problem for optimal DF strategy involves the construction of two codes: $\mathcal{X}_1$ of rate R_1 and $\tilde{\mathcal{X}}$ of rate R. While the relay's codebook $\mathcal{X}_1$ can be constructed as a conventional error-correcting code that guarantees successful decoding at the destination, the source's codebook $\tilde{\mathcal{X}}$ must be constructed so that it can be decoded at *both* the relay and the destination.

The SNR at which the source-relay code must work is denoted by SNR_+ and the SNR at which the source-destination code must work is denoted by SNR_-. All channels are assumed to be Gaussian, hence, for Gaussian inputs, the capacities of the source-relay link and the source-destination link are

$$R_+ = \frac{1}{2}\log(1+\mathrm{SNR}_+) \qquad \textbf{6.1}$$

$$R_- = \frac{1}{2}\log(1+\mathrm{SNR}_-) \qquad \textbf{6.2}$$

respectively. The overall DF rate in terms of R_+ and R_- becomes

$$R = \min\{R_+, R_1 + R_-\}. \qquad \textbf{6.3}$$

Throughout this chapter we assume that the source-relay link is reliable, and, the extra parity bits generated by the relay are given to the destination "in hand" (i.e. without using a different codebook).

6.2 Bilayer Expurgated LDPC Codes

Let $\mathcal{C}$ be a linear $(n, n-k_1)$ LDPC code of rate $\frac{(n-k_1)}{n}$. The codebook $\tilde{\mathcal{X}}$ should be a capacity-approaching code for the source-relay channel at SNR_+ with a rate R_+. Let k_2 be the number of extra parity bits on the source codeword $\mathbf{x}$, generated by the relay and forwarded to the destination. Then, a sub-code of $\mathcal{C}$ which satisfies two sets of parities: k_1 zero parities enforced by the source's codebook and k_2 presumably non-zero parity bits provided by the relay, should form an $(n, n-(k_1+k_2))$ capacity approaching code for decoding at the destination, i.e., at SNR_- with a rate R_-. Note that the performance of a practical bilayer code is characterized by two gaps to the capacities at SNR_+ and at SNR_-.

The decoding of the sub-code of $\mathcal{C}$ with the extra k_2 presumably non-zero parity bits can be performed in the same way as the decoding of a conventional LDPC code. A graphical representation follows.

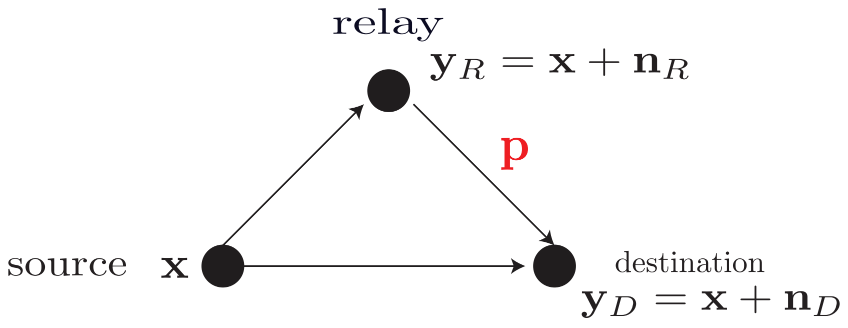

relay
$\mathbf{y}_R = \mathbf{x} + \mathbf{n}_R$
$\mathbf{p}$
source
$\mathbf{x}$
destination
$\mathbf{y}_D = \mathbf{x} + \mathbf{n}_D$

Figure 6.2: Parity-forwarding scheme for the relay channel.

To be more clear:

1. For the source-relay link, $\mathbf{x} \in \mathcal{C}$, with parity-check matrix $\mathbf{H}$ such that $\mathbf{Hx} = \mathbf{0}$.
2. rate($\mathcal{C}$)$\leq$ rate(source $\rightarrow$ relay). Hence, relay decodes $\mathbf{x}$.
3. rate($\mathcal{C}$)$>$ rate(source $\rightarrow$ destination). Hence, destination cannot decode $\mathbf{x}$.
4. The parity-check matrix of $\mathcal{C}_{\text{bilayer}}$ denoted by $\mathbf{H}_{\text{bilayer}}$ is given by

$$\mathbf{H}_{\text{bilayer}} = \begin{bmatrix} \mathbf{H} \\ \mathbf{H}_1 \end{bmatrix},$$

where $\mathbf{H}_1$ is the upper layer of the bilayer graph. In the general case

$$\mathbf{H}_{\text{bilayer}} \cdot \mathbf{x} = \begin{bmatrix} \mathbf{0} \\ \mathbf{p} \end{bmatrix},$$

and $\mathbf{p}$ can be computed as $\mathbf{H}_1 \hat{\mathbf{x}}_R$.

5. Finally, rate($\mathcal{C}_{\text{bilayer}}$)$\leq$ rate(source $\rightarrow$ destination) and, therefore, destination decodes $\mathbf{x}$ using $\mathbf{H}_{\text{bilayer}}$ and parity vector $\mathbf{p}$ sent by the relay.

The proposed LDPC code structure is shown in **Fig. 6.3**. We call the proposed code structure *bilayer-expurgated* LDPC code [7], as the overall graph represents an expurgated sub-code of the lower layer code. The first (lower) layer corresponds to an $(n, n-k_1)$ code and the second (upper) layer consists of the k_2 extra parity bits which modify the first layer in a way that the resulting $(n, n-(k_1+k_2))$ sub-code represented by the overall graph is suitable for the source-destination channel.

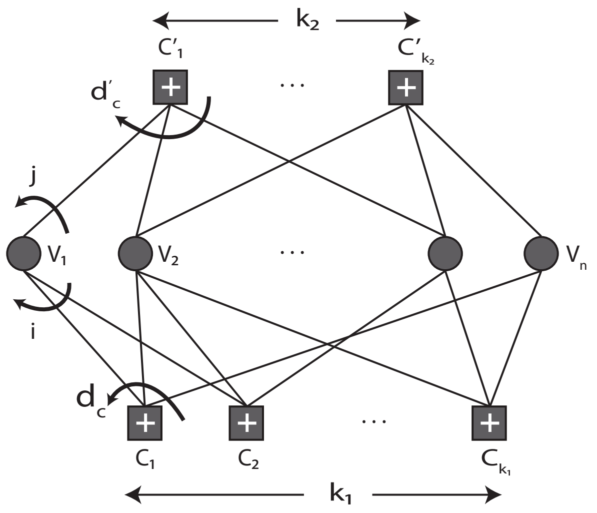


Figure 6.3: The bilayer expurgated code. The lower sub-graph represents an LDPC code for source-relay channel. The overall graph represents an LDPC code for the destination.

Bilayer-Expurgated LDPC Code Ensemble

An ensemble of bilayer-expurgated LDPC codes is defined as follows. The bilayer graph of the code, consists of three sets of nodes and two sets of edges. The three sets of nodes correspond to one set of variable nodes and two sets of check nodes: the lower check nodes, corresponding to the check nodes in the lower sub-graph of **Fig. 6.3**, and the upper check nodes, corresponding to the check nodes in the upper sub-graph in **Fig. 6.3**. Edges are grouped in two sets: those connecting the variable nodes to the lower check nodes, and those connecting the variable nodes to the upper check nodes. We call an edge a lower edge if it connects a variable to a lower check node. Similarly, an upper edge denotes an edge belonging to the upper sub-graph in **Fig. 6.3**.

The lower degree of a variable node is defined as the number of lower edges connected to it. Likewise, the upper degree of a variable node is defined as the number of upper edges connected to it. Notice that the minimum lower variable degree is 2, since, the lower sub-graph should be a valid LDPC code for the source-relay channel. The minimum upper variable degree is 0, since the bilayer graph is simply the vertical concatenation between the valid LDPC code for the source-relay and the upper layer. Hence, some variable nodes may not participate in any of the k_2 extra parity checks generated by the relay. A variable node is said to have degree (i, j) if it has a lower degree i and an upper degree j. Similarly, an edge is of degree (i, j) if it is connected to a degree (i, j) variable node.

We assume regular check degrees for both lower and upper check nodes. The lower check degree is denoted by d_c and the upper check degree is denoted by $d_c^{'}$. The ensemble of bilayer LDPC codes can be characterised by a variable degree distribution

$$\lambda_{i,j},\ i \geq 2,\ j \geq 0 \quad \text{with} \sum_{i\geq 2, j\geq 0} \lambda_{i,j} = 1,$$

which defines the percentage of edges with lower degree i and upper degree j and a parameter $0 < \eta = \frac{d_c \cdot k_1}{d_c k_1 + d_c^{'} k_2} < 1$ which defines the percentage of lower edges in the bilayer graph.

Bilayer Density Evolution

Because the ensemble of bilayer-expurgated LDPC codes is different from a conventional LDPC code ensemble, conventional density evolution cannot be applied. This gives rise to bilayer density evolution [7]. The evolution of two densities should be tracked: the lower density corresponding to the density of messages in the lower sub-graph, and the upper density corresponding to the density of the messages in the upper sub-graph.

Let p^{ℓ} and q^{ℓ} denote the message probability density functions (pdf) at the input of the lower and upper check nodes at ℓ-th decoding iteration. Let p'^{ℓ} and q'^{ℓ} denote the evolved versions of p^ℓ and q^ℓ after the check updates[1]. Then, the density-evolution update at a degree (i,j) variable node can be computed as follows:

$$p_{i,j}^{(\ell+1)} = (\otimes^{i-1} p'^{\ell}) \otimes (\otimes^{j} q'^{\ell}) \otimes p_c,\ i \geq 2, j \geq 0 \qquad \textbf{6.4}$$

$$q_{i,j}^{(\ell+1)} = (\otimes^{i} p'^{\ell}) \otimes (\otimes^{j-1} q'^{\ell}) \otimes p_c,\ i \geq 2, j \geq 0 \qquad \textbf{6.5}$$

where p_c denotes the density of the log-likelihood ratio received from the channel, and $\otimes^i$ denotes convolution of order i.

Averaging over $\lambda_{i,j}$, the input message densities to the lower and upper check nodes at the $(\ell+1)$-th iteration are computed as follows:

$$p^{(\ell+1)} = \sum_{i\geq 2, j\geq 0} \frac{i}{i+j} \lambda_{i,j} p_{i,j}^{(\ell+1)}, \qquad \textbf{6.6}$$

$$q^{(\ell+1)} = \sum_{i\geq 2, j\geq 0} \frac{j}{i+j} \lambda_{i,j} q_{i,j}^{(\ell+1)}, \qquad \textbf{6.7}$$

where $\frac{i}{i+j}$ is the probability that a degree (i,j) edge is a lower edge and $\frac{j}{i+j}$ is the probability that a degree (i,j) edge is an upper edge.

The lower-graph degree (i,j) error profile function $e_{i,j}^{1}(p^\ell, q^\ell)$ is defined as the message error probability corresponding to the density $p_{i,j}^{(\ell+1)}$ after one density evolution iteration with input message densities p^ℓ and q^ℓ. The aforementioned error probability can be computed using [1, equation (4.54), p.201]. Likewise, $e_{i,j}^{2}(p^\ell, q^\ell)$ is defined as the message error probability corresponding to the density $q_{i,j}^{(\ell+1)}$ after one density evolution iteration with input message densities p^ℓ and q^ℓ. Hence, the overall message error probability at $(\ell+1)$-th decoding iteration $e(p^{(\ell+1)}, q^{(\ell+1)})$ can be computed as:

$$e(p^{(\ell+1)}, q^{(\ell+1)}) = \sum_{i\geq 2, j\geq 0} \lambda_{i,j} \Big(\frac{i}{i+j} e_{i,j}^{1}(p^\ell, q^\ell) + \frac{j}{i+j} e_{i,j}^{2}(p^\ell, q^\ell) \Big) \qquad \textbf{6.8}$$

Bilayer-Expurgated LDPC Code Optimization

The design of a bilayer-expurgated LDPC code involves finding a variable degree distribution $\lambda_{i,j}$, $i \geq 2$, $j \geq 0$, a parameter η, and a pair of check degree d_c and d_c' such that the lower sub-graph represents a capacity-approaching LDPC code over the source-relay channel at SNR_+, and the overall bilayer code is capacity approaching at $SNR_- < SNR_+$. The design approach is to fix the lower graph code to be an optimal capacity-approaching

[1] p'^{ℓ} and q'^{ℓ} can be computed using the conventional density evolution check node rule described in [6].

LDPC code at SNR_+ and search for a variable degree distribution $\lambda_{i,j}$ that is consistent with the lower graph code and is capacity-approaching at SNR_-.

By fixing the lower graph, i.e., fixing n, k_1 and the lower variable degree distribution λ_i, the rate of the bilayer code, defined by $1 - \frac{(k_1+k_2)}{n}$ can be maximized by minimizing k_2 or equivalently maximizing η. The lower degree distribution λ_i is related to $\lambda_{i,j}$ as follows:

$$\lambda_i = \sum_{j\geq 0} \frac{i}{i+j}\lambda_{i,j}. \qquad \textbf{6.9}$$

For fixed λ_i, the aforementioned equation can be written as

$$\sum_{j\geq 0} \frac{i}{i+j}\lambda_{i,j} - \eta\lambda_i = 0. \qquad \textbf{6.10}$$

Hence, the linear program can be written as:

$$\begin{aligned} \max_{\lambda_i,\eta,j} \quad & \eta \\ \text{subject to} \quad & \sum_{j\geq 0} \frac{i}{i+j}\lambda_{i,j} - \eta\lambda_i = 0, \quad i \geq 2, \\ & \sum_{i\geq 2, j\geq 0} \lambda_{i,j}\Big(\frac{i}{i+j}e^1_{i,j}(p^\ell,q^\ell) + \frac{j}{i+j}e^2_{i,j}(p^\ell,q^\ell)\Big) < \mu^h e(p^\ell,q^\ell),\ \ell = 1,\cdots,L \\ & \sum_{i\geq 2, j\geq 0} \lambda_{i,j} = 1 \end{aligned}$$

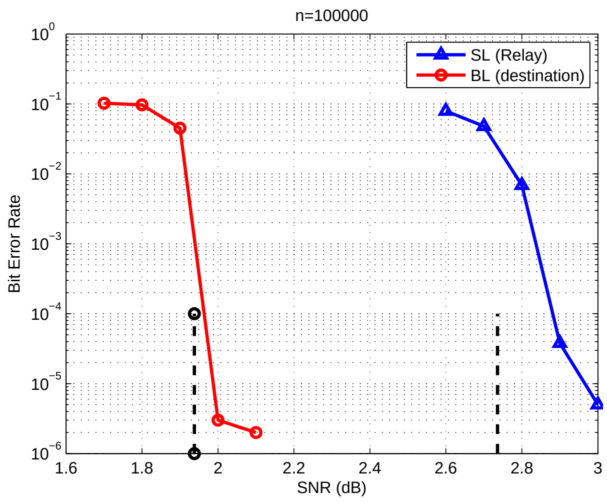
where h is the optimization iteration number and ℓ is the decoding iteration number.

To initialize the above iterative optimization, i.e., to find a good initialisation degree distribution, we solve the following linear programming problem:

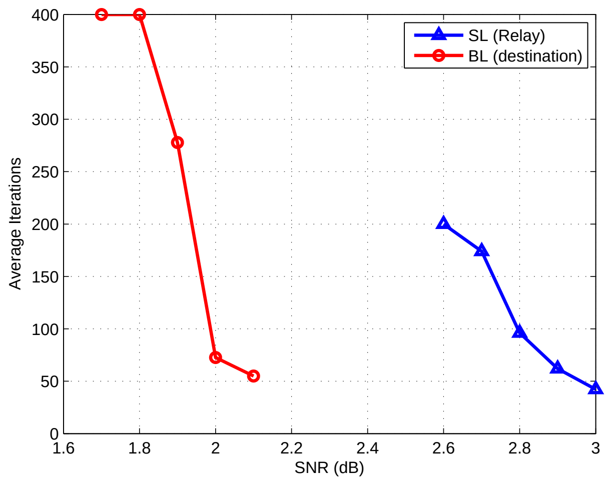
$$\begin{aligned} \min_{\lambda_i,\eta,j} \quad & \eta \\ \text{subject to} \quad & \sum_{i\geq 2, j\geq 0} \lambda_{i,j} = 1 \\ & \sum_{j\geq 0} \frac{i}{i+j}\lambda_{i,j} - \eta\lambda_i = 0, \quad i \geq 2 \end{aligned}$$

which will result in an initial code with maximum number of parity checks k_2, which ensures quick decoding convergence. We mention that when the gap between SNR_+ and SNR_- is small, the difference between the optimal d_c and d_c' is likely to be small, and the aforementioned scheme works well. For the case where the gap is large, we introduce in the following subsection the *bilayer-lengthened* LDPC codes.

Finally, we present a simulation under all-zero codeword assumption, using $\lambda_{i,j}$ degree distribution of code A illustrated in Table I in [7]. Note that since we assumed that the relay decodes reliably and that we send the *all-zero* codeword, it follows that *all* parity bits and therefore the extra k_2 parity bits must be 0.



(a) Bit Error Rate



(b) Average number of iterations

Figure 6.4: Bit Error Rate of a bilayer-expurgated LDPC code. Dashed lines represent the theoretical limits.

6.3 Bilayer Lengthened LDPC Codes

We now propose a second coding structure for DF based on code lengthening, which is designed to address the problem of the expurgated structure for large SNR differences. Lengthening of a linear code refers to the process of increasing the codeword length while keeping the number of parity-check equations fixed. **Fig. 6.5** depicts a bilayer-lengthened LDPC code [7] in which the overall graph corresponds to a lengthened version of the lower code. Designing a bilayer-lengthened code corresponds to finding an overall graph so that the lower graph corresponds to a good LDPC code at rate R_- optimized for SNR_-, while the overall bilayer graph, which can be constructed by adding extra variable nodes to the lower graph, represents a good LDPC code at rate R_+ optimized for SNR_+. Being capacity-approaching at two different rates is a core feature of this code.

The source encodes its data using the bilayer LDPC code corresponding to the overall graph. Thus, each codeword satisfies all parity-check nodes present in the bilayer graph in contrast to the expurgated scheme where the source encodes its data over the lower sub-graph. The relay first decodes the source codeword over the bilayer graph. It then helps the destination by sending the values of upper n_2 variable nodes in the next block using the following scheme. The relay generates a set of $k_2 = (1 - R_{-})n_2$ extra parity bits for the upper variable nodes, using the parity-check matrix of a separate conventional LDPC code $\mathcal{C}_2$ of rate R_{-} optimized for SNR_{-}. The relay forwards these presumably non-zero extra parity bits to the destination using another code of rate R_1. The destination first decodes the set of k_2 extra parity bits for upper n_2 variable bits provided by the relay. These k_2 parity bits are used to decode the upper n_2 variable nodes of the source codeword, which are then removed from the graph, reducing the resulting code's rate and thus allowing the destination to decode the remaining n_1 variable nodes.

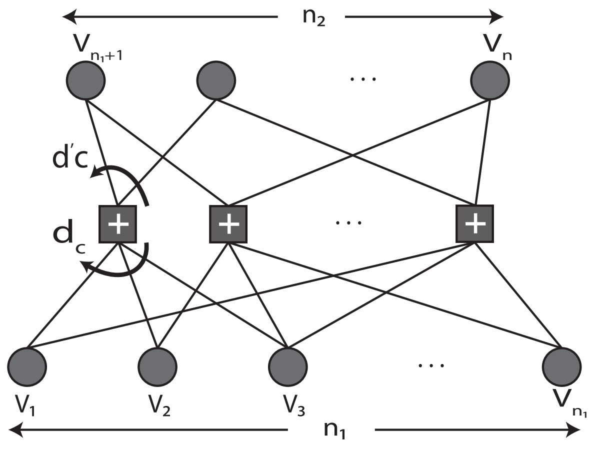


Figure 6.5: The bilayer-lengthened code. The relay decodes the overall graph and provides the value of upper variable nodes to the destination. The destination decodes the lower sub-graph.

Observe that, when the destination removes the upper n_2 variable nodes, the value of the parity-check nodes in the graph must be updated. That is, consider the leftmost check node of **Fig. 6.5**. Since the relay works reliably, i.e., all parity-check equations of the bilayer graph have even-parity, it follows that $V_1 \oplus V_2 \oplus V_3 \oplus V_{n_1+1} \oplus V_{n_1+2} = 0$. After removing the upper n_2 variable nodes, the aforementioned equation becomes $V_1 \oplus V_2 \oplus V_3 = V_{n_1+1} \oplus V_{n_1+2}$. Under the aforementioned assumptions, the bilayer-lengthened construction can be done using two conventional LDPC constructions which correspond to the lower graph and upper graph and then we have to concatenate (row-wise) the two parity-check matrices in order to construct the bilayer graph (code).

Let us define the bilayer-lengthened LDPC code ensemble. The nodes are grouped into one set of check nodes and two sets of variable nodes: the lower variable nodes corresponding to the lower graph and the upper variable nodes corresponding to the upper graph. The edges are grouped into two sets: those connecting the check nodes to the lower variable nodes, and those connecting check nodes to the upper variable nodes. As in the expurgated case we assume regular check degrees.

The ensemble of bilayer-lengthened LDPC codes is defined by the lower variable degree distribution $\lambda_i^1, i \geq 2$, which corresponds to the probability that a lower edge in connected to a degree i variable node, the upper variable degree distribution $\lambda_i^2, i \geq 2$, which corresponds to the probability that an upper edge is connected to a degree i variable node. The lower and upper distributions λ_i^1 and λ_i^2 satisfy $\sum_{i \geq 2} \lambda_i^1 = 1$, and $\sum_{i \geq 2} \lambda_i^2 = 1$.

The ensemble of bilayer-lengthened LDPC codes is *not* equivalent to conventional LDPC codes, because in a conventional LDPC code there is *only one* variable degree distribution. This gives rise to bilayer density evolution since conventional density evolution is not valid.

Bilayer Density Evolution

It is clear that in order to predict the performance of an infinite-length bilayer-lengthened LDPC code, we need to track the evolutions of two densities in the upper and lower sub-graphs of the lengthened graph. Let p^ℓ and q^ℓ denote the message densities in the lower and upper parts of the graph at the beginning of the ℓ-th decoding iteration. Let $p_\ell^{'}$ and $q_\ell^{'}$ denote the evolved versions of p^ℓ and q^ℓ after check node updates. $p_\ell^{'}$ and $q_\ell^{'}$ can be computed as follows:

$$p_\ell^{'} = \left(\star^{d_c - 1} p^\ell\right) \star \left(\star^{d_c^{'}} q^\ell\right) \qquad \textbf{6.11}$$

$$q_\ell^{'} = \left(\star^{d_c} p^\ell\right) \star \left(\star^{d_c^{'} - 1} q^\ell\right) \qquad \textbf{6.12}$$

where $\star$ is the convolution operator as described in **5.15**.

Let $p_i^{(\ell+1)}(q_i^{(\ell+1)})$ denote the output message density after a variable update at a variable node of degree i in the lower(upper) sub-graph, with an input message density $p'^{\ell}(q'^{\ell})$.

Then we have:

$$p_i^{(\ell+1)} = \otimes^{i-1} p'^{\ell} \otimes p_c, \;\; i \geq 2 \tag{6.13}$$

$$q_i^{(\ell+1)} = \otimes^{i-1} q'^{\ell} \otimes p_c, \;\; i \geq 2, \tag{6.14}$$

where p_c is the channel message density.
The message densities in the lower and upper sub-graphs after the variable update rule at the beginning of $(\ell+1)$-th iteration, $p^{(\ell+1)}$ and $q^{(\ell+1)}$, can be computed as follows:

$$p^{(\ell+1)} = \sum_{i \geq 2} \lambda_i^1 p_i^{(\ell+1)} \tag{6.15}$$

$$q^{(\ell+1)} = \sum_{i \geq 2} \lambda_i^2 q_i^{(\ell+1)}. \tag{6.16}$$

Let $e(p^{(\ell+1)}, q^{(\ell+1)})$ denote the message error probability of the message densities $p^{(\ell+1)}$ and $q^{(\ell+1)}$ at the beginning of the $(\ell+1)$-th decoding iteration. Let $e_i^1(p^\ell, q^\ell)$ denote the message error probability corresponding to $p_i^{(\ell+1)}$, which is the message density of degree-i lower nodes after one density evolution iteration with input message densities p^ℓ and q^ℓ. Similarly, let $e_i^2(p^\ell, q^\ell)$ denote the message error probability corresponding to $q_i^{(\ell+1)}$, which is the message density of degree-i upper nodes after one density evolution iteration with input message densities p^ℓ and q^ℓ. The overall message error probability at the beginning of the $(\ell+1)$-th iteration, $e(p^{(\ell+1)}, q^{(\ell+1)})$, can be computed [7] as

$$e(p^{(\ell+1)}, q^{(\ell+1)}) = \sum_{i \geq 2} \eta \lambda_i^1 e_i^1(p^\ell, q^\ell) + (1-\eta) \lambda_i^2 e_i^2(p^\ell, q^\ell), \tag{6.17}$$

where $\eta = \frac{d_c}{d_c + d_c'}$ denotes the percentage of lower edges in the bilayer-lengthened graph.

Bilayer-Lengthened LDPC code optimization

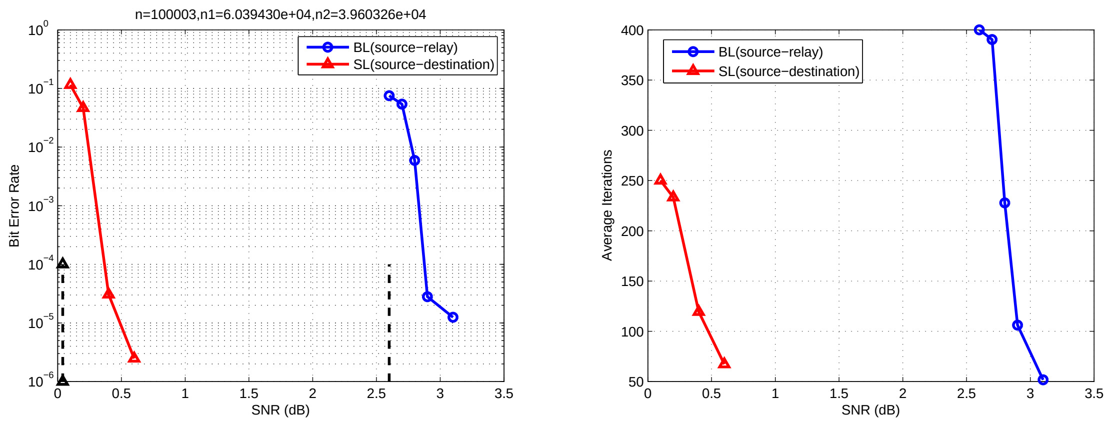
The design of a bilayer-lengthened LDPC code involves finding a pair of variable degree distributions λ_i^1 and λ_i^2 $(i \geq 2)$ and a pair of check degrees d_c and d_c' for the lower and upper sub-graphs in the bilayer structure of **Fig. 6.5**, such that the overall graph is a capacity-approaching LDPC code for a Gaussian channel at SNR_+ while the lower graph is a capacity-approaching LDPC code at SNR_-.
Similar to the previous design we fix the check degrees d_c and d_c', we also fix the lower variable degree distribution λ_i^1 to be a capacity-approaching distribution for a conventional LDPC code optimized at SNR_- (which is found independently). The design problem is

now reduced to finding an upper variable-degree distribution λ_i^2 such that the overall lengthened graph is a capacity-approaching code at SNR_+. Observe that, in contrast to the expurgated case, the lower rate is fixed here and the higher rate code is optimized. The rate of the bilayer graph is $1 - \frac{k}{n_1+n_2}$, where k is the number of check nodes. According to **3.14**, the number of upper variable nodes n_2 is given by $d_c' \cdot k \cdot \sum_{i\geq 2} \frac{\lambda_i^2}{i}$. Thus, fixing the lower graph code and d_c', the rate of the overall code can be maximized by maximizing $\sum_{i\geq 2} \frac{\lambda_i^2}{i}$. Fixing η, d_c and d_c', the linear programming update for λ_i^2 can be formulated as follows:

$$\begin{aligned} \max_{\lambda_i^2} \quad & \sum_{i\geq 2} \frac{\lambda_i^2}{i} \\ \text{subject to} \quad & \sum_{i\geq 2} \eta\lambda_i^1 e_i^1(p^\ell, q^\ell) + (1-\eta)\lambda_i^2 e_i^2(p^\ell, q^\ell) \\ & < \mu^h e(p^\ell, q^\ell),\ \ell = 1, \cdots, L \\ & \sum_{i\geq 2} \lambda_i^2 = 1 \end{aligned}$$

where h denotes the optimization iteration round, and ℓ is the decoding iteration number. As an initialisation, $\lambda^2_{\max(d_v)} = 1$ is used. This code class covers the cases where the gap between SNR_+ and SNR_- is large. Thus, expurgated and lengthened bilayer LDPC codes are complementary code structures which cover the entire range of SNRs.

Finally, we present a simulation, under all-zero codeword assumption, using λ_i^1 and λ_i^2 degree distributions of code F illustrated in Table II in [7].



(a) Bit Error Rate (b) Average number of iterations

Figure 6.6: Bit Error Rate of a bilayer-lengthened LDPC code. Dashed lines represent the theoretical limits.

6.4 Gaussian Approximation for Bilayer Codes

Since bilayer density evolution has high computational complexity, the authors in [17] propose a Gaussian approximation for bilayer-expurgated codes. We denote the mean of message updates at the ℓ-th iteration at the input of lower (upper) check nodes by $\mu_{\mathrm{v}}^{1,\ell}(\mu_{\mathrm{v}}^{2,\ell})$. In the same fashion, we denote the mean of message updates at the output of lower (upper) check nodes by $\mu_u^{1,\ell}(\mu_u^{2,\ell})$. Using the message update rule at a degree (i,j) variable node we have

$$\begin{aligned}
\mu_{\mathrm{v},i,j}^{1,\ell} &= \mu_{u_0} + (i-1)\mu_{u,i,j}^{1,(\ell-1)} + j\cdot\mu_{u,i,j}^{2,(\ell-1)}\\
\mu_{\mathrm{v},i,j}^{2,\ell} &= \mu_{u_0} + i\cdot\mu_{u,i,j}^{1,(\ell-1)} + (j-1)\cdot\mu_{u,i,j}^{2,(\ell-1)}
\end{aligned}$$

where μ_{u_0} denotes the channel LLR mean. Working with the mean message updates at the variable nodes, we can obtain as in [12, equation (14)],

$$\begin{aligned}
\mu_{\mathrm{v}}^{1,\ell} &= \phi\Bigg(\mu_{u_0} + (i-1)\phi^{-1}\Big(1-[1-\mu_{\mathrm{v}}^{1,(\ell-1)}]^{(d_c-1)}\Big)\\
&\quad + j\phi^{-1}\Big(1-[1-\mu_{\mathrm{v}}^{2,(\ell-1)}]^{d_c'-1}\Big)\Bigg)\\
&= g\Big(\mu_{\mathrm{v}}^{1,(\ell-1)},\mu_{\mathrm{v}}^{2,(\ell-1)}\Big),\\
\mu_{\mathrm{v}}^{2,\ell} &= \phi\Bigg(\mu_{u_0} + (j-1)\phi^{-1}\Big(1-[1-\mu_{\mathrm{v}}^{2,(\ell-1)}]^{(d_c'-1)}\Big)\\
&\quad + i\phi^{-1}\Big(1-[1-\mu_{\mathrm{v}}^{1,(\ell-1)}]^{d_c-1}\Big)\Bigg)\\
&= h\Big(\mu_{\mathrm{v}}^{1,(\ell-1)},\mu_{\mathrm{v}}^{2,(\ell-1)}\Big).
\end{aligned}$$

The overall mean of message updates at variable nodes [12] can then be written as

$$\begin{aligned}
\mu_{\mathrm{v}}^{\ell} &= \sum_{i\geq 2, j\geq 0}\lambda_{i,j}\left(\frac{i}{i+j}\mu_{\mathrm{v}}^{1,\ell} + \frac{j}{i+j}\mu_{\mathrm{v}}^{2,\ell}\right)\\
&= \sum_{i\geq 2, j\geq 0}\lambda_{i,j}\left(\frac{i}{i+j}g\Big(\mu_{\mathrm{v}}^{1,(\ell-1)},\mu_{\mathrm{v}}^{2,(\ell-1)}\Big) + \frac{j}{i+j}h\Big(\mu_{\mathrm{v}}^{1,(\ell-1)},\mu_{\mathrm{v}}^{2,(\ell-1)}\Big)\right).
\end{aligned}$$

We can also calculate μ_{v}^{ℓ} using η as

$$\mu_{\mathrm{v}}^{\ell} = \eta\cdot\mu_{\mathrm{v}}^{1,\ell} + (1-\eta)\cdot\mu_{\mathrm{v}}^{2,\ell}.$$

According to [12], a necessary condition to obtain successful decoding is given by

$$\begin{aligned}&\sum_{i\geq 2, j\geq 0} \lambda_{i,j}\left(\frac{i}{i+j} g\left(\mu_{\mathrm{v}}^{1,(\ell-1)}, \mu_{\mathrm{v}}^{2,(\ell-1)}\right) + \frac{j}{i+j} h\left(\mu_{\mathrm{v}}^{1,(\ell-1)}, \mu_{\mathrm{v}}^{2,(\ell-1)}\right)\right)\\ &< \eta \cdot \mu_{\mathrm{v}}^{1,\ell} + (1-\eta)\cdot \mu_{\mathrm{v}}^{2,\ell}.\end{aligned}$$

The aforementioned expression is approximately linear in $\lambda_{i,j}$ and this allows for linear optimization programming. The goal is to maximize the rate of the lower graph, that is,

$$\begin{aligned}\max_{i} \quad & \sum_{i\geq 2} \frac{\lambda_i}{i}\\ \text{subject to} \quad & \sum_{i\geq 2} \lambda_i \phi\left(\mu_{u_0} + (i-1)\phi^{-1}\left(1-[1-\mu_{\mathrm{v}}]^{d_c-1}\right)\right) < \mu_k \mu_{\mathrm{v}}^{\ell},\\ & \sum_{i\geq 2, j\geq 0} \lambda_{i,j}\left(\frac{i}{i+j} g\left(\mu_{\mathrm{v}}^{1,(\ell-1)}, \mu_{\mathrm{v}}^{2,(\ell-1)}\right) + \frac{j}{i+j} h\left(\mu_{\mathrm{v}}^{1,(\ell-1)}, \mu_{\mathrm{v}}^{2,(\ell-1)}\right)\right)\\ & < \eta \cdot \mu_{\mathrm{v}}^{1,\ell} + (1-\eta)\cdot \mu_{\mathrm{v}}^{2,\ell},\\ & \lambda_i = \frac{1}{\eta} \sum_{j\geq 0} \frac{i}{i+j} \lambda_{i,j}.\end{aligned}$$

Naturally, the codes designed using the Gaussian approximation perform slightly worse than those designed using density evolution but there is a complexity trade-off.

Chapter 7

Construction of LDPC Codes

After designing a code, which in fact means finding a good degree distribution pair (λ, ρ) for the specific channel, we have to construct a code from the optimised ensemble. It is vital to choose a good graph or equivalently a good parity-check matrix representation of the ensemble in order to achieve the best possible performance. Various constructions exist, each having its pros and cons. We will present Gallager codes, the historically first construction, the configuration model [1] which we use to construct all the codes of this thesis, and concluding with the construction through progressive edge growth algorithm which avoids cycles and optimize the girth of the graph to be as large as possible. In addition, we will extend the use of configuration model to Bilayer LDPC codes which we use to construct parity-check matrices for the relay channel.

7.1 Gallager Construction

This is the original method for constructing regular LDPC codes introduced by Gallager in [4].

The goal is to construct a (n, j, k) parity-check matrix, where n is the number of columns, j is the number of 1's in each column and k is the number of ones in each row. Since, in a Tanner graph the number of edges in each side must be the same, it follows that the number of rows of the parity-check matrix is $\frac{n \cdot j}{k}$. Hence, the rate r of the code is given by $r \geq 1 - \frac{j}{k}$ with equality if and only if the parity-check matrix is full-rank.

In order to construct an ensemble of (n, j, k) matrices we proceed as follows:

- The matrix $\mathcal{H}$ is divided into j sub-matrices each of which has $\frac{n}{k}$ rows. Namely,

$$\mathcal{H} \triangleq \begin{bmatrix} \mathcal{H}_1 \\ \vdots \\ \mathcal{H}_j \end{bmatrix}.$$

- Let $\mathcal{H}_1$ denotes the first of the sub-matrices. It contains all its 1's in descending order. That is, the i-th row of $\mathcal{H}_1$ contains 1's in columns $(i-1)k+1$ to ik.

- The other sub-matrices can be constructed by merely column permutations of $\mathcal{H}_1$.

To clarify the aforementioned construction procedure we illustrate the following example:

Example 7.1. Assume that $n = 20$, $j = 3$ and $k = 4$. Therefore, the parity-check matrix $\mathcal{H}$ will be divided into 3 sub-matrices, namely

$$\mathcal{H} \triangleq \begin{bmatrix} \mathcal{H}_1 \\ \mathcal{H}_2 \\ \mathcal{H}_3 \end{bmatrix}.$$

Each of the sub-matrices will have $\frac{20}{4}$ rows. The i-th row of $\mathcal{H}_1$ will have 1's in columns $(i-1)\cdot 4+1$ to $4i$. Consequently,

$$\mathcal{H}_1 = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \end{bmatrix}.$$

To construct $\mathcal{H}$ we need two random permutations of $\mathcal{H}_1$ say

$$\mathcal{H}_2 = \begin{bmatrix} 0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&1&1&0&0&1&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&0&0 \\ 1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0 \end{bmatrix}.$$

and

$$\mathcal{H}_3 = \begin{bmatrix} 0&0&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&1&0&0 \\ 1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&1&1&0&0&0 \\ 0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0 \end{bmatrix}.$$

By concatenating the 3 matrices we constructed $\mathcal{H}$. Note that as the block length increases, the probability that any two columns are equal or linearly dependent decreases. Thus, the probability that the actual rate is close to the design rate increases. This construction is rather simple, but it gives no guarantee for the rank of the resulting parity-check matrix, it does not avoid small cycles and it can only create regular codes. Another drawback of this method is that we can not create codes that are encodable with linear time complexity. $\diamond$

7.2 Configuration Model

Recall that in § 3.3 we showed how to sample from an ensemble. In this section, we will give an extra graphical representation of the construction procedure. It is crucial to transform a given degree distribution pair (λ, ρ) from an edge perspective to the corresponding degree distribution pair (L, R) from a normalized node perspective. We remind that the coefficients L_i of the latter distribution pair can be calculated as

$$L_i = \frac{1}{\sum_j \frac{\lambda_j}{j}} \frac{\lambda_i}{i}, \text{ for } i = 2, \dots, \ell_{\max}. \tag{7.1}$$

Using L_i we can calculate the number of variable nodes per degree and the corresponding number of edges, since, L_i represents the probability of a variable node be of degree i. Namely,

$$\#\text{variable\_nodes\_per\_degree\_}i = L_i \cdot n$$
$$\#\,\text{of\_edges\_connected\_to\_variable\_nodes\_of\_degree\_}i = L_i \cdot n \cdot \text{degree\_}i$$

where n denotes the block-length of the code and degree_i the corresponding variable node degree. After computing the aforementioned quantities we can proceed to the construction procedure as follows:

- "Generate" n variable nodes and m check nodes with their sockets.

- Label the variable and check node sockets separately with the set $[\Lambda'(1)] = \{1, \ldots, \sum_i i\Lambda_i\}$.

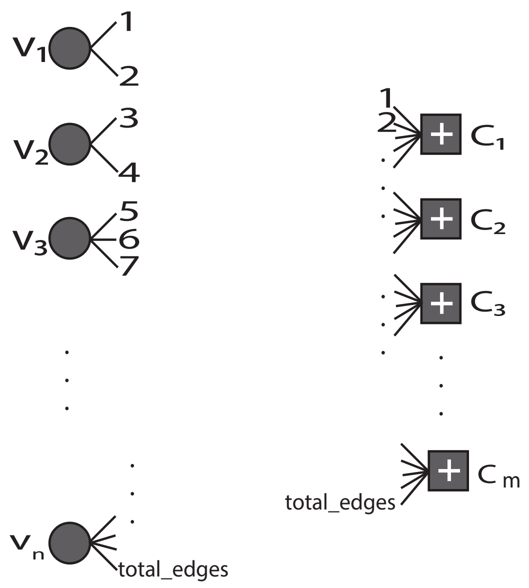


Figure 7.1: Labeled Tanner graph of a concentrated irregular LDPC code.

total_ edges is the last socket (edge) which is equal to the total number of edges ($= \sum_i i\Lambda_i$).

- Pick a permutation π on $[\Lambda'(1)]$ at random with uniform probability from the set of all $\big(\sum_i i\Lambda_i\big)!$ such permutations.

- Let π be a permutation on $[\Lambda'(1)]$. Connect i-th variable node edge socket with $\pi(i)$-th check node edge socket.

- Put "1" in the appropriate position of $\mathcal{H}$. That is, if i-th check node is connected to the j-th variable node an *odd* number of times put 1 in (i, j) position of the parity check matrix. Otherwise, put 0.

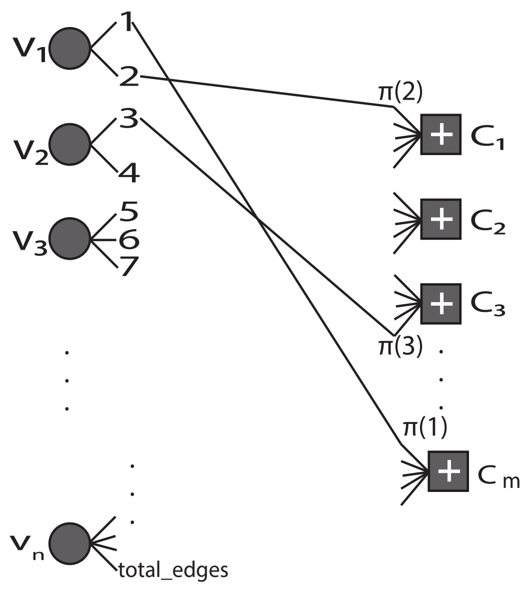


Figure 7.2: Connection procedure of a concentrated irregular LDPC code.

- The last step is to try to remove short-length cycles to boost code's performance.

7.3 Progressive Edge Growth (PEG) Algorithm

As we have seen, the constructions containing cycles in randomly generated graphs, have a probability which tends to 0 as the block length tends to infinity. However, for a code to be practical, its length can not be arbitrarily large. For short or moderately lengthed LDPC codes (up to, say, some thousand bits), the unfavourable constructs have relatively high probability, thus having non-negligible impact on the code's decoding performance. Our goal is to construct a graph having as large girth as possible, given the code length and degree distribution, which is a rather hard combinatorial problem. However, a suboptimal yet simple and well performing algorithm was proposed in [16].

The basic idea is pretty straightforward: the graph is constructed in an edge-by-edge manner with each edge placed so that it has the minimum possible impact on the overall graph girth. This means that, fundamentally, an edge is placed between the variable node in question and the most distant check node in the graph. In the optimal case where the distance is infinite, i.e. no path exists between the variable node and the check node, the new edge creates no additional cycles.

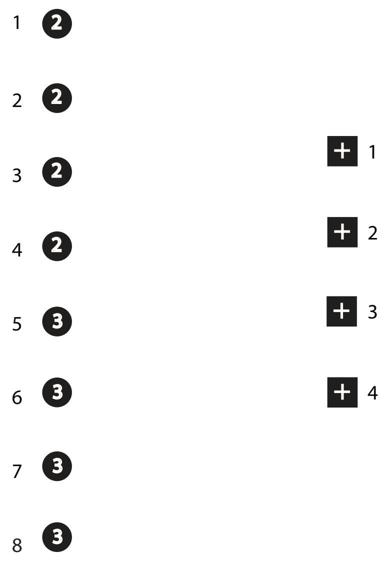
Given the number of variable nodes, the number of check nodes and the variable node degree distribution, this algorithm assigns degrees to each variable node according to the degree distribution and calculates the degree distribution of the check nodes, making it as $uniform$ as possible.
Before describing the algorithm, we will explain the notation used. c_i and s_i denote the i-th check and variable node respectively. E_{s_j} denotes the set containing all edges incident to variable node s_j and $E_{s_j}^k$ denotes the k-th edge incident to s_j. $\mathcal{N}_{s_j}^{\ell}$ is the *neighbourhood* of variable node s_j at depth ℓ and is defined as the set containing all check nodes reached by a sub-graph spreading from variable node s_j within depth ℓ. $\overline{\mathcal{N}}_{s_j}^{\ell}$ denotes the complement of $\mathcal{N}_{s_j}^{\ell}$,i.e., the st containing all check nodes which are not reached by a sub-graph spreading from variable node s_j within depth ℓ.

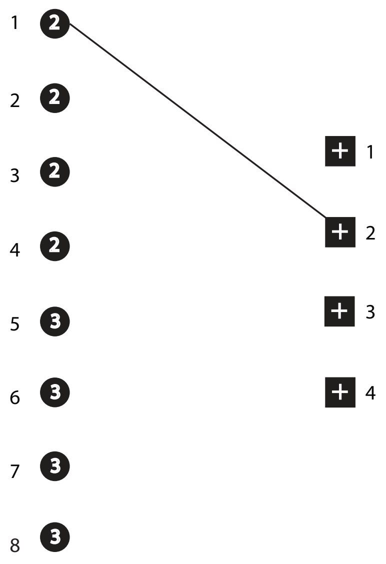
Algorithm 2 PEG Algorithm

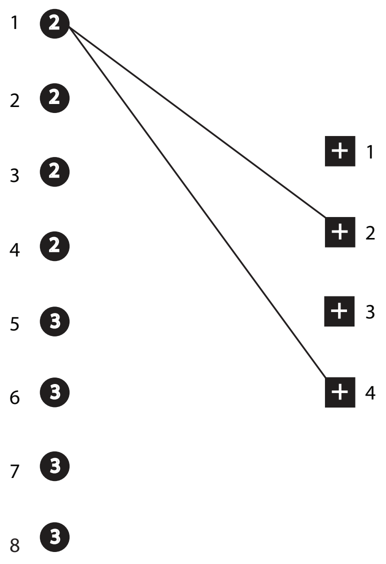
```
1: procedure PEG($\lambda$,$n$,$m$)        ▷ $\lambda := \lambda(x)$, $n :=$ block-length, $m :=$ # check nodes
2:     for $j = 0$ to $n-1$ do
3:         for $k = 0$ to $d_{s_j} - 1$ do
4:             if $k = 0$ then
5:                 $E_{s_j}^0 \leftarrow \text{edge}(c_i, s_j)$, where $E_{s_j}^0$ is the first edge incident to $s_j$ and $c_i$
6:                 is a check node such that it has the lowest check-node degree
7:                 under the current graph setting $E_{s_0} \cup E_{s_1} \cup \cdots \cup E_{s_j - 1}$ .
8:             else
9:                 expand a sub-graph from variable node $s_j$ up to depth $\ell$ under the
10:                current graph setting such that the cardinality of $\mathcal{N}_{s_j}^{\ell}$ stops increasing
11:                but is less than $m$, or $\overline{\mathcal{N}}_{s_j}^{\ell} \neq \emptyset$ but $\overline{\mathcal{N}}_{s_j}^{(\ell+1)} = \emptyset$, then $E_{s_j}^k \leftarrow \text{edge}(c_i, s_j)$,
12:                where $E_{s_j}^k$ is the $k$-th edge incident to $s_j$ and $c_i$ is a check node picked
13:                from the set $\overline{\mathcal{N}}_{s_j}^{\ell}$ having the lowest check-node degree.
14:            end if
15:        end for
16:    end for
17: end procedure
```

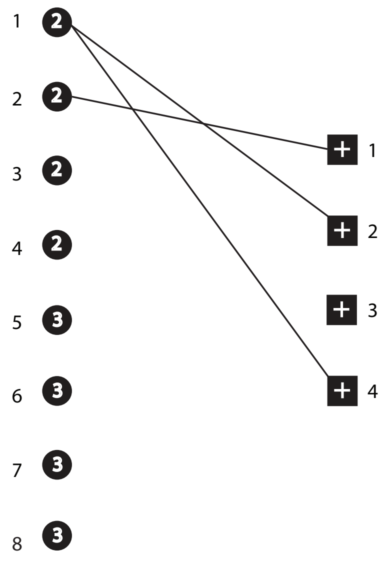
To clarify the PEG algorithm, we illustrate an example in an edge-by-edge manner.

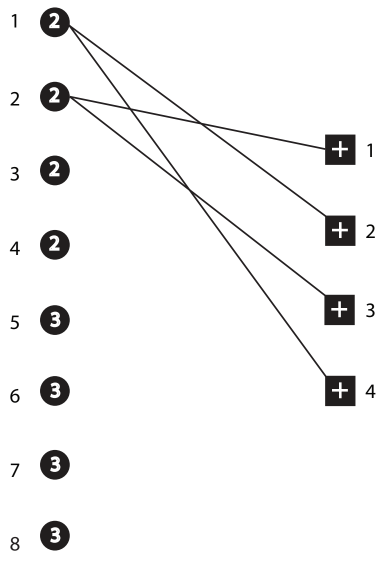
Example 7.2. Consider a Tanner graph with $L(x) = 0.5x^2 + 0.5x^3$. Numbers in variable nodes refer to number of edges in each node. As mentioned earlier, the goal is to make the graph have as large girth as possible.

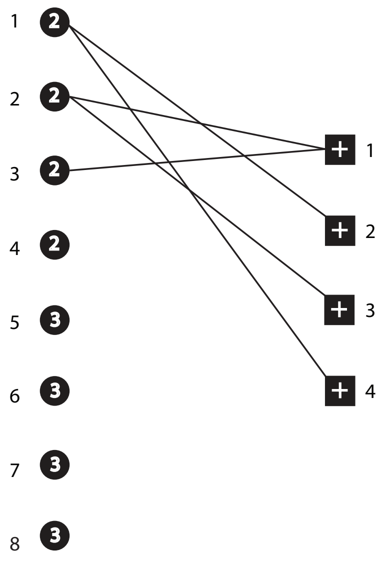

(a) Initial unconnected graph

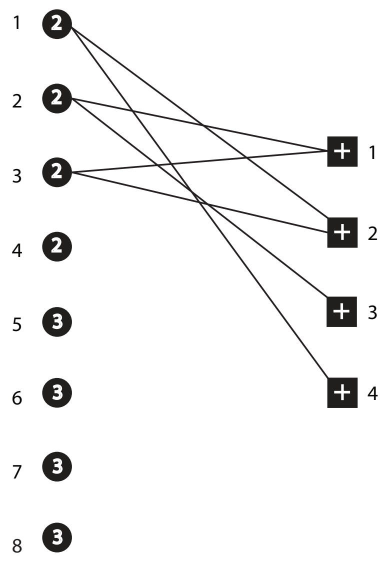

(b) Step 1


(c) Step 2

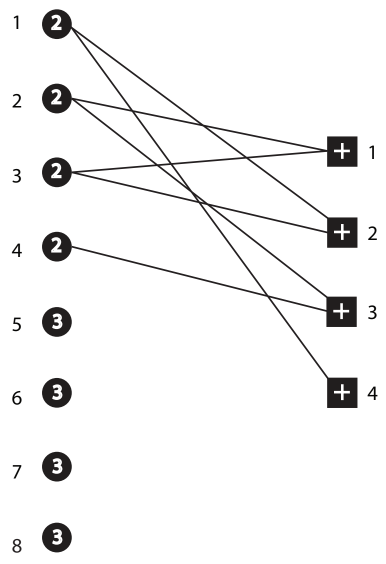

(d) Step 3


(e) Step 4

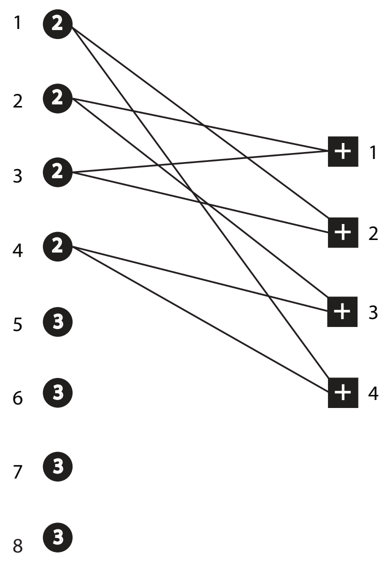

(f) Step 5



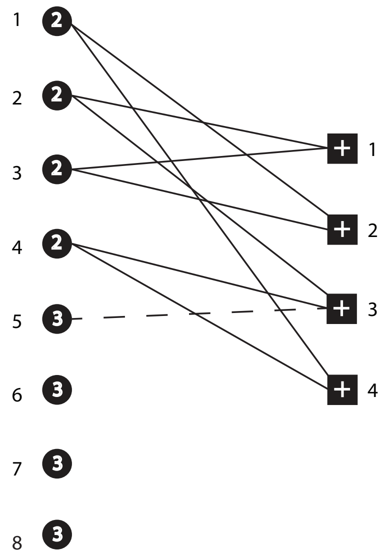
(g) Step 6



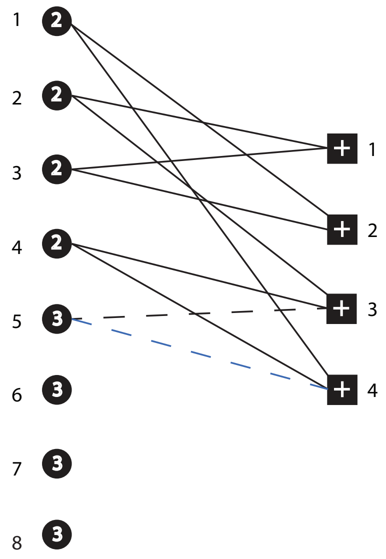
(h) Step 7



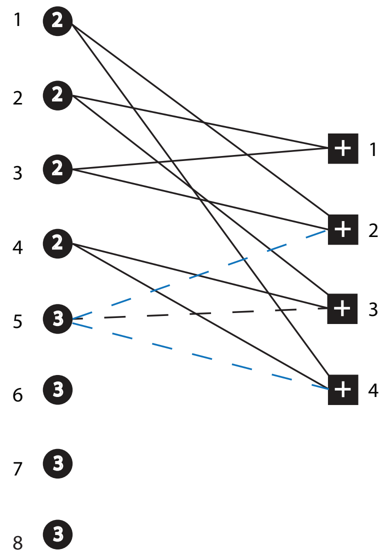
(i) Step 8



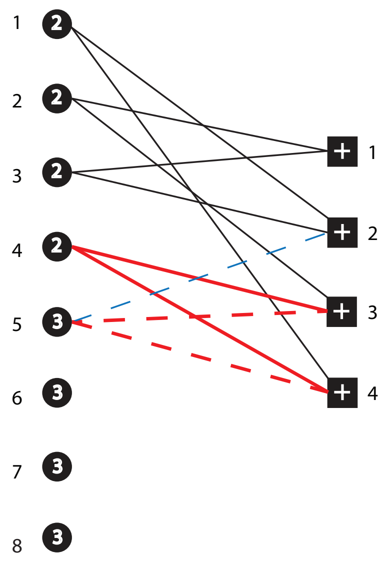
(j) Step 9

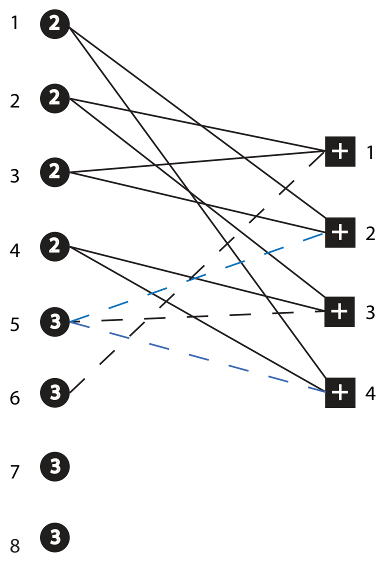


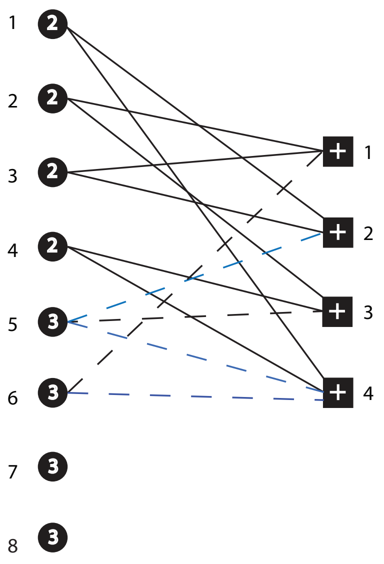
(k) Step 10

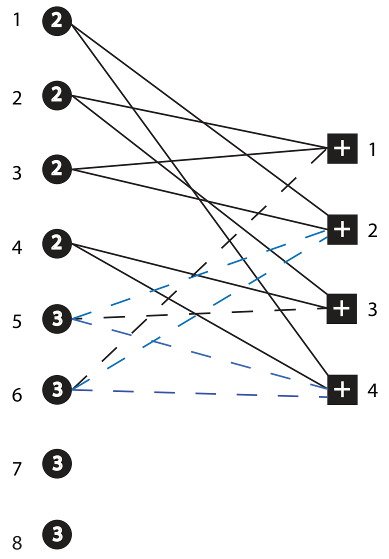


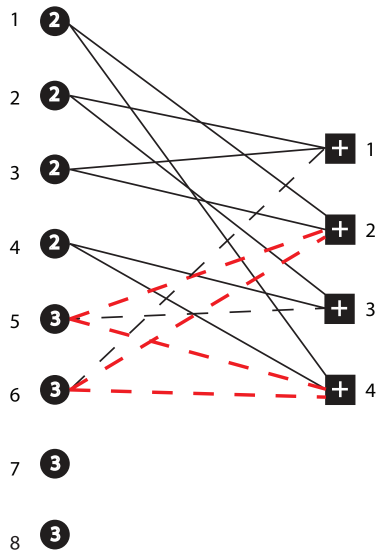
(l) Step 11


(m) Step 12


(n) Step 13


(o) Step 14


(p) Step 15


(q) Step 16

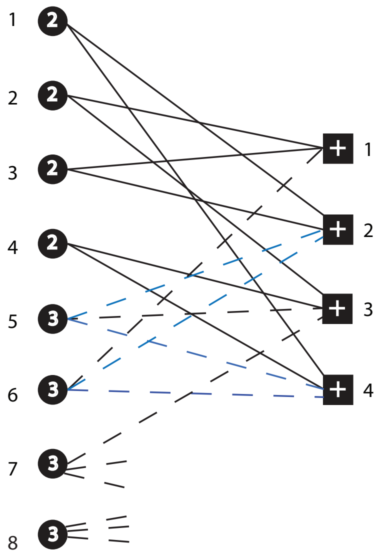

(r) Step 17

Figure 7.3: Edge-by-edge derivation of a Tanner graph using PEG algorithm. Broken arrows represent edges emanating from variable nodes with degree 3. Edges in red represent a cycle of length 4. Blue broken arrows needs more intuition to choose the appropriate (most distant) check nodes.

We omit the last steps for the reader. Notice that in our example we couldn't avoid length four cycle. PEG algorithm for a $(3, 6)$ regular LDPC ensemble is optimal, i.e., for moderate block-lengths the parity-check matrix of a $(3,6)$ LDPC code has usually girth 10 or 12. Obviously PEG is used and for irregular ensembles. $\diamond$

Without going into details, using a slight modification [16], PEG algorithm can produce codes which are in upper triangular form, possessing the very pleasant property of having linear time encoding. This modification only affects the $(m-1)$ first variable nodes, since they correspond to the $m \times m$ sub-matrix of $\mathcal{H}_{m \times n}$ which we want to be upper triangular. For the remaining variable nodes, the unmodified PEG algorithm is used. The modification is quite simple, we only allow connections which result in 1's over the diagonal line of the first $m \times m$ sub-matrix while making sure that the diagonal line has strictly non-zero elements, so that the $\mathcal{H}_{1:m,1:m}$ is guaranteed to have full rank.
Below we present an example in order to compare the two last mentioned constructions.

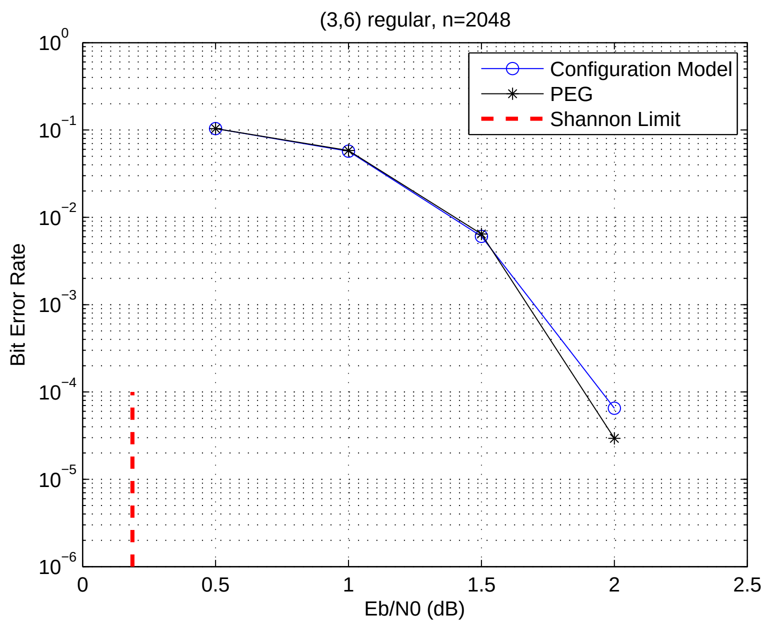


Figure 7.4: Bit Error Rate of a (3,6) regular LDPC code using configuration model and PEG algorithm.

7.4 Configuration Model for Bilayer Construction

In this section, we present an extension of § 7.2 which will be used for the construction of bilayer-graphs. At first, let us consider the construction of a bilayer-expurgated graph depicted in **Fig. 6.3**. After the optimization procedure $\lambda_{i,j}$ table is known. As for the single-user graph it is crucial to transform $\lambda_{i,j}$ to $L_{i,j}$ which represents the probability of a variable node be of lower degree i and upper degree j. An extension of the equation we derived from L_i is as follows:

$$L_{i,j} = \frac{1}{\sum_k \sum_\ell \frac{\lambda_{k,\ell}}{k+\ell}} \cdot \frac{\lambda_{i,j}}{i+j}, \quad i \geq 2,\ j \geq 0. \tag{7.2}$$

Bilayer-Expurgated Construction

Recall that the distribution λ_i is related to $\lambda_{i,j}$ as follows:

$$\lambda_i = \frac{1}{\eta} \sum_{j \geq 0} \frac{i}{i+j} \lambda_{i,j}.$$

Hence, given λ_i and d_c, the source-relay parity-check matrix, which corresponds to the lower graph of the bilayer graph, can be constructed as a conventional LDPC code explained in § 7.2.

The non-trivial part of the bilayer construction is to construct the upper layer in such a way so that to be compatible when we concatenate the two parity-check matrices of the two layers. The upper layer construction can be done as follows:

1. After computing $L_{i,j}$ table, we can calculate the number of *upper* edges with respect to lower degrees i and upper degrees j. Namely, using a Matlab like notation, upper_edges(i,j)=$\lceil\ n \cdot L_{i,j} \cdot j\ \rceil$ for $i = 2, \ldots, \ell_{\max}$ and $j = 0, \ldots, r_{\max}$ where $\ell_{\max}$ and $r_{\max}$ are the maximum lower and upper degrees.

2. The number of variable nodes with respect to lower degree i and upper degree j for each $i \geq 2, j \geq 0$, can be computed as $\lceil L_{i,j} \cdot n \rceil$, where n is the block-length of the code.

3. For each (fixed) i we find the number of upper edges. That is, using a Matlab like notation again, we have i_num_of_upper_edges(i)= sum(upper_edges(i,:)). In other words, we find the number of upper edges corresponding to each lower degree i.

4. We repeat the previous step for variable all nodes and therefore we have i_num_of_variable_nodes(i)=sum(num_of_variable_nodes(i,:)).

5. Taking the summation of i_num_of_variable_nodes and i_num_of_upper_edges we can find the total number of variable nodes and upper edges respectively.

6. The extra k_2 check nodes can be computed as sum(i_num_of_upper_edges)/d_c' where d_c' is the upper degree of check nodes.

7. Using the aforementioned steps, we are able to associate properly the upper edges to the corresponding variable nodes using exactly the same way as in § 7.2.

Therefore, by concatenating the matrices of the two layers, we create the bilayer parity-check matrix.

Bilayer-Lengthened Construction

The bilayer-lengthened construction is quite straightforward since we have to construct one parity-check matrix for the lower graph, which has n_1 variable nodes and k check-nodes with a rate R_-, and a second parity-check matrix for the upper layer with n_2 variable nodes and k check nodes. Then the bilayer graph can be constructed by row-wise concatenation between the two matrices. The variable degree distributions for lower and upper graphs are λ_i^1 and λ_i^2 respectively. Because lower graph and bilayer graph have the same number of check nodes it follows that

$$n_1 = n\frac{1 - R_-}{1 - R_+}. \quad \textbf{7.3}$$

Hence, given k, d_c and λ_i^1 we can construct the lower graph as in § 7.2. Also, given k, d_c' and λ_i^2 we compute the upper variable nodes n_2 as $d_c' \cdot k \sum_{i \geq 2} \frac{\lambda_i^2}{i}$ and then we are able to construct the upper graph with the same way as the lower one. Finally, we construct the bilayer graph as mentioned earlier.

Chapter 8

Conclusion

We saw that LDPC codes can be designed to perform close to the capacity of different types of channels. They possess several distinct advantages. First, belief-propagation decoding for LDPC codes have linear, with the code length, complexity for fixed number of iterations. Second, they have an easily understood graphical representation which is helpful for their analysis. Furthermore, we presented an effective construction method for picking LDPC codes at random from LDPC ensembles and we extended it for Bilayer LDPC codes. Finally, the main design tool, Density Evolution, which predicts the asymptotic performance of a belief-propagation decoder is also presented for both LDPC and Bilayer LDPC codes.

Bibliography

[1] Richardson, T., & Urbanke, R. L. (2008). Modern coding theory. Cambridge University Press.

[2] C. E. Shannon. A mathematical theory of communication. Bell Sys. Tech. J., 27:379-423 and 623-656, July/ Oct. 1948.

[3] S.-Y.Chung,"On the construction of some capacity-approaching coding schemes", Ph.D. dissertation, MIT, Cambridge, MA, 2000.

[4] R. G. Gallager, "Low-density parity-check codes", IRE Transactions on Information Theory, vol. 8, no. 1, pp. 21-28, Jan. 1962.

[5] Richardson, T. J., & Urbanke, R. L. (2001). The capacity of low-density parity-check codes under message-passing decoding. Information Theory, IEEE Transactions on, 47(2), 599-618.

[6] Richardson, T. J., Shokrollahi, M. A., & Urbanke, R. L. (2001). Design of capacity-approaching irregular low-density parity-check codes. Information Theory, IEEE Transactions on, 47(2), 619-637.

[7] Razaghi, P., & Yu, W. (2007). Bilayer low-density parity-check codes for decode-and-forward in relay channels. Information Theory, IEEE Transactions on, 53(10), 3723-3739.

[8] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., & Spielman, D. A. (2001). Improved low-density parity-check codes using irregular graphs. Information Theory, IEEE Transactions on, 47(2), 585-598.

[9] M.Ardakani,”Efficient Analysis, Design and Decoding of Low-Density Parity-Check Codes”, Ph.D. dissertation, University of Toronto, 2004.

[10] Wymeersch, Henk. Iterative receiver design. Vol. 234. Cambridge: Cambridge University Press, 2007.

[11] Chung, S. Y., Forney Jr, G. D., Richardson, T. J., & Urbanke, R. (2001). On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. Communications Letters, IEEE, 5(2), 58-60.

[12] Chung, S. Y., Richardson, T. J., & Urbanke, R. L. (2001). Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. Information Theory, IEEE Transactions on, 47(2), 657-670.

[13] Johnson, S. J. (2009). Iterative error correction: turbo, low-density parity-check and repeat-accumulate codes. Cambridge University Press.

[14] Tanner, R. M. (1981). A recursive approach to low complexity codes. Information Theory, IEEE Transactions on, 27(5), 533-547.

[15] Chakrabarti, Arnab, et al. ”Low density parity check codes for the relay channel.” Selected Areas in Communications, IEEE Journal on 25.2 (2007): 280-291.

[16] Hu, X. Y., Eleftheriou, E., & Arnold, D. M. (2005). Regular and irregular progressive edge-growth tanner graphs. Information Theory, IEEE Transactions on, 51(1), 386-398.

[17] Cances, J. P., & Meghdadi, V. (2009). Optimized low density parity check codes designs for half duplex relay channels. Wireless Communications, IEEE Transactions on, 8(7), 3390-3395.

[18] E. C. van der Meulen, ”Three-terminal communication channels,” Advanced Applied Probability, vol. 3, pp. 120-154, 1971.

[19] T. M. Cover and A. A. E. Gamal, ”Capacity theorems for the relay channel”, IEEE Trans. Inf. Theory, vol.25, no.5, pp. 572-584, Sept. 1979.

[20] Ryan, W., & Lin, S. (2009). Channel codes: classical and modern. Cambridge University Press.

[21] MacKay, D. J., & Neal, R. M. (1996). Near Shannon limit performance of low density parity check codes. Electronics letters, 32(18), 1645-1646.

[22] Sipser, M., & Spielman, D. A. (1994, November). Expander codes. In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (pp. 566-576). IEEE Comput. Soc. Press.

[23] Moon, T. K. (2005). Error correction coding. Mathematical Methods and Algorithms. Jhon Wiley and Son.

[24] Costello Jr, D. J., Dolecek, L., Fuja, T. E., Kliewer, J., Mitchell, D. G., & Smarandache, R. (2013). Spatially Coupled Sparse Codes on Graphs-Theory and Practice. arXiv preprint arXiv:1310.3724.

[25] Optional B-LDPC coding for OFDMA PHY, IEEE Std. IEEE 802.16e-04/78, 2004.