AALBORG UNIVERSITET

Message Passing Algorithm and Linear Programming Decoding for LDPC and Linear Block Codes

Institute of Electronic Systems

Signal and Information Processing in Communications

Nana Traore · Shashi Kant · Tobias Lindstrøm Jensen

The Faculty of Engineering and Science

Aalborg University

9th Semester

TITLE:

Message Passing Algortihm and Linear Programming Decoding for LDPC and Linear Block Codes

PROJECT PERIOD:

P9,
4^{th} September, 2006
- 4^{th} January, 2007

PROJECT GROUP:

976

GROUP MEMBERS:

Nana Traore
Shashi Kant
Tobias Lindstrøm Jensen

SUPERVISORS:

Ingmar Land
Joachim Dahl
External:
Lars Kristensen
from Rohde-Schwarz

NUMBER OF COPIES: 7

REPORT PAGE COUNT: 100

APPENDIX PAGE COUNT: 6

TOTAL PAGE COUNT: 106

ABSTRACT:

Two iterative decoding methods, message passing algorithm (MPA) and linear programming (LP) decoding, are studied and explained for an arbitrary LDPC and binary linear block codes.
The MPA, sum-product and max-product/min-sum algorithm, perform local decoding operations to compute the marginal function from the global code constraint defined by the parity check matrix of the code. These local operations are studied and the algorithm is exemplified.
The LP decoding is based on a LP relaxation. An alternative formulation of the LP decoding problem is explained and proved. An improved LP decoding method with better error correcting performance is studied and exemplified.
The performance of two methods are also compared under the BEC.

Preface

This 9^{th} semester report serves as a documentation for the project work of the group 976 in the period from 4^{th} September, 2006 to 4^{th} January, 2007. It is to comply with the demands at Aalborg University for the SIPCom specialization at 9^{th} semester with the theme "Systems and Networks".

Structure

The report is divided into a number of chapters whose contents are outlined here.

- ”Introduction” part contains the introduction of the project and the problem scope.
- ”System Model” describes which model that is considered in the report along with assumptions and different types of decoding.
- The chapter ”Binary Linear Block Codes” describes linear block codes and explain the concept of factor graphs and LDPC codes.
- In ”Message Passing Algorithm” different message passing algorithms are given along with examples. Performance for decoding in the BEC is proven and exemplified.
- ”Linear Programming Decoding” is about the formulation of the decoding problem as a linear programming problem. Interpretations of two different formulations are given and decoding in the BEC is examined. A possible improvement of the method is also described using an example.
- In the chapter ”Comparison” the two decoding methods message passing algorithm and linear programming decoding are compared.
- ”Conclusion” summarizes everything and points out the results.

Reading Guidelines

The chapters ”System Model” and ”Binary Linear Block Codes” are considered as background information for the rest of the report. If the reader is familiar with these concepts he/she could skip these chapters and still understand the report.

Nomenclature

References to literature are denoted by brackets as [] and literature may also contain a reference to a specific page. The number in the brackets refers to the bibliography which can be found at the back of the main report on the page 99 . Reference to figures (and tables) are denoted by ”Figure/Table x.y” and equations by ”equation (x.y)” where x is a chapter number and y is a counting variable for the corresponding element in the chapter.

A vector is denoted by bold face ”$\boldsymbol{a}$” and matrix by ”$\underline{\mathbf{A}}$”, always capital. Stochastic processes are in upper case ”X”, deterministic process are in lower case ”x”, but giving reference to the ”same” variable. From section to section it is considered whether it is more convenient to consider the variable as stochastic or deterministic. Be aware of the differences between the use of the subscripts in ”$\boldsymbol{x}_1$” which means the first vector and ”x_1” which means the first symbol in the vector $\boldsymbol{x}$.

The words in the *italic* form are used in the text to accentuate the matter.

Enclosed Material

A CD ROM containing MATLAB source code is enclosed in the back of the report. Furthermore, a Postscript, DVI and PDF version of this report are also included in the CD ROM. A version with hyperlinks is also included in DVI and PDF.

Nana Traore

Shashi Kant

Tobias Lindstrøm Jensen

Contents

Introduction 1

The channel coding is a rudimentary technique to transmit digital data reliably over a noisy channel. If a user wants to send information reliably using a mobile phone to another user, then the information will be transmitted through some channel to the another user. However, the channel like the air is considered to be unreliable because the transmission path varies, noise is introduced and there is also interference from the other users. So, if the transmitted data is [001] and the received data is [011] which is shown in the Figure 1.1, then the received data was corrupted by the noisy channel. What could be the solution to cope with these problems?

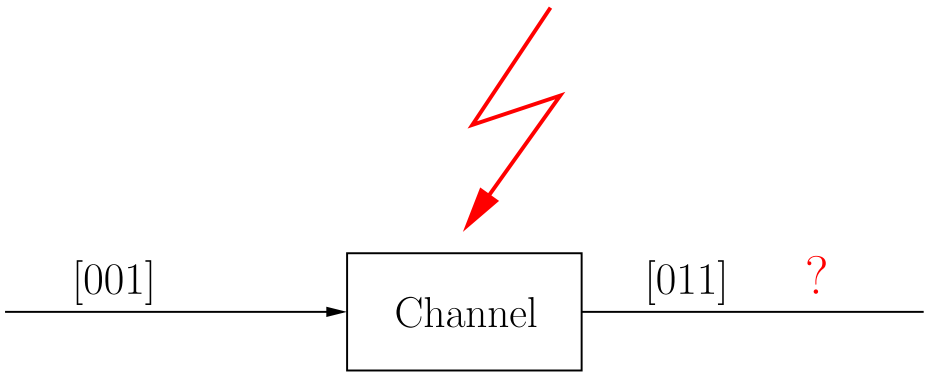


Figure 1.1: *Data transmitted and received across a channel without channel coding.*

The solution to combat the noisy channel is channel coding so as to transmit and receive the information reliably. In other words, the channel coding is a technique which introduces the redundancy in the information before the transmission of the information across a channel. This redundancy is exploited at the receiver in order to detect or correct the errors introduced by the noisy channel.

Figure 1.2 shows a basic channel coding system. For example, a repetition code can be chosen having length 3. It means that the repetition code will transmit every bit of an information 3 times. If the *source* transmits the information word say $[001]$, then the encoded data will be $[000\ 000\ 111]$. Therefore, the redundancy is introduced by the *encoder*. If this encoded data or codeword is sent across a noisy channel which may flip some of the bits, then say the received word may be $[010\ 001\ 101]$. Thereafter, this received word is sent to the *decoder* which estimates the information word $[001]$ by exploiting the redundancy. So, if there are more 0s than 1s in the received word, then the decoder will estimate the information word as 0 otherwise 1. Using this scheme, as long as no more than 1 out of every block of 3 bits are flipped, the original information can be recovered. Thus, the information from the source can be transmitted successfully across the channel to the receiver's *sink*.

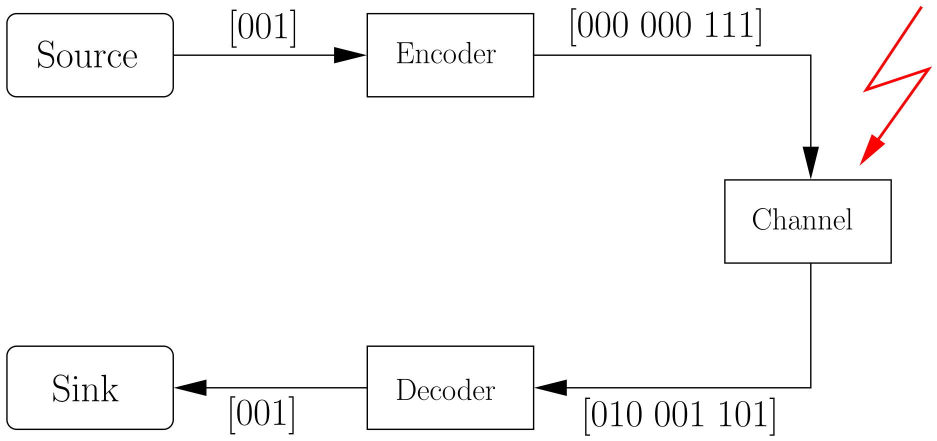


Figure 1.2: *The basic channel coding model.*

The channel codes are also known as *error-correcting codes*. The study of channel codes began with the pioneered work of Hamming [7] and Shannon [17].

In this project, two iterative methods *message passing algorithm* (MPA) and *linear programming* (LP) *decoding* are studied and compared theoretically for the decoding of LDPC (Low Density Parity Check) codes. However, simple linear binary block codes are usually considered to explain the concepts behind these iterative decoding methods. The LDPC codes are used as a motivation to be decoded by these two methods. LDPC codes were invented in 1960 by R. Gallager [6] but the LDPC codes were obscured before the advent of Turbo-codes in 1993. After the invention of Turbo-codes [2], the LDPC codes are now one of the intensely studied area in coding. It's also been proved recently that the LDPC codes have outperformed Turbo-codes while keeping lower complexity [15]. The decoding of the LDPC codes are highly intensive research arena. These are the motivations behind studying the decoding of LDPC codes using iterative techniques.

System Model 2

In this chapter, we consider the system model and different channel models that are used through out this report. Moreover, we introduce commonly used assumptions and definitions that are heavily used in the following chapters.

2.1 Basic System Model

Consider now the system showed in Figure 2.1.

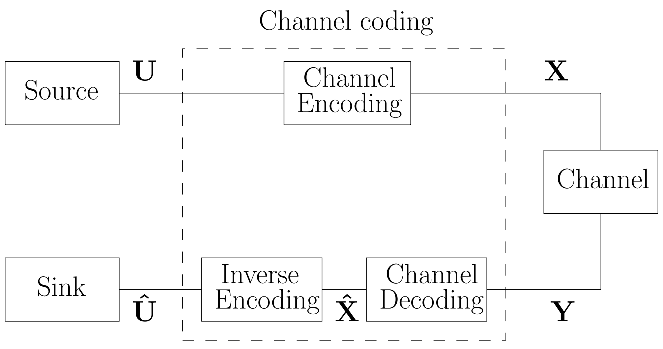


Figure 2.1: *The basic system model considered in this report.*

Where, the following definitions are used. All vectors will be column vectors, if it is not specified whether it is column or row vector. The assumptions about the random variables are described in section 2.3.

- *Information word*: $\boldsymbol{U} = [U_1\ U_2\ \dots\ U_K]^T \in \mathbb{F}_2^K$
- *Codeword*: $\boldsymbol{X} = [X_1\ X_2\ \dots\ X_N]^T \in \mathcal{C}$
  Where, $\boldsymbol{X} \in \mathbb{F}_2^N$ and $\mathcal{C}$ is the set of all possible (legal) codewords $|\mathcal{C}| = 2^K$, defined as $\mathcal{C} = \{\boldsymbol{X} : \boldsymbol{X} = \text{enc}(\boldsymbol{U}); \boldsymbol{U} \in \mathbb{F}_2^K\}$
- *Received word*: $\boldsymbol{Y} = [Y_1\ Y_2\ \dots\ Y_N]^T$; Domain of $\boldsymbol{Y}$ depends on the channel used
- *Estimated codeword*: $\hat{\boldsymbol{X}} = [\hat{X}_1\ \hat{X}_2\ \dots\ \hat{X}_N]^T \in \mathbb{F}_2^N$

- *Estimated information word*: $\hat{\boldsymbol{U}} = [\hat{U}_1\ \hat{U}_2\ \dots\ \hat{U}_K]^T \in \mathbb{F}_2^K$
- *Code rate*: $R = K/N$

In this report, only binary codes are considered. The domain of $\boldsymbol{Y}$ is defined by the channel used. In the following section 2.2, the different channels and the corresponding output alphabet for $\boldsymbol{Y}$ are described.

Considering Figure 2.1, the information words are generated from the (stochastic) source. The information words are one-to-one mapped (encoded) to a codeword in the set $\mathcal{C}$ depending on the information word. The codeword is transmitted across the channel. The received word is decoded to the estimated codeword $\hat{\boldsymbol{X}}$ and mapped to an information word as $\hat{\boldsymbol{U}} = \text{enc}^{-1}(\hat{\boldsymbol{X}})$. A successful transmission is when $\hat{\boldsymbol{U}} = \boldsymbol{U}$ or $\hat{\boldsymbol{X}} = \boldsymbol{X}$. Random variables are in upper case, deterministic variables are in lower case, but giving reference to the ”same” variable. From section to section it is considered whether it is more convenient to consider the variable as random or deterministic.

2.2 Channels

In this report, we consider the following three channels, the *Binary Symmetric Channel (BSC)*, the *Binary Erasure Channel (BEC)* and the *Binary Input Additive White Gaussian Noise Channel (BI-AWGNC).*

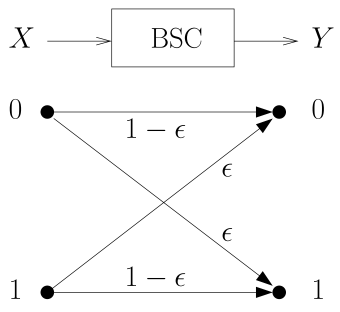


Figure 2.2: *The BSC with cross over probability ϵ.*

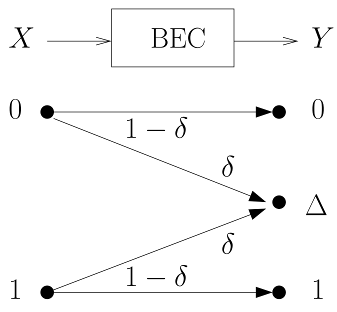


Figure 2.3: *The BEC with erasure probability δ.*

For the BSC in Figure 2.2 some input values are with probability ϵ flipped at the output, $Y \in \{0, 1\}$. In the BEC Figure 2.3 the input values can be erased (Δ) with probability δ, mapping X into $Y \in \{0, \Delta, 1\}$.

For the (normalized) BI-AWGNC in Figure 2.4, the input X is mapped into $X' \in \{+1, -1\}$ which is added with Gaussian white noise, resulting in the output $Y = X' + W$, where $W \sim \mathcal{N}(0, N_0/2E_s)$. The conditional distribution of Y is

$$\Pr(\,y \,|\, x'\,) = \Pr{}_W(y - x') = \frac{1}{\sqrt{2\pi(N_0/2E_s)}} \exp\left(-\frac{(y - x')^2}{(N_0/E_s)}\right) \qquad (2.1)$$

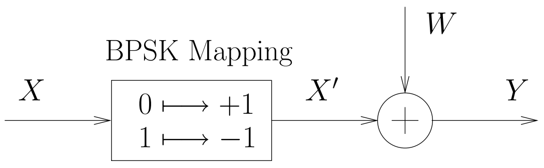


Figure 2.4: *The normalized BI-AWGNC.*

The output alphabet for the BI-AWGNC is $y \in \mathbb{R}$. The *Signal to Noise Ratio (SNR)* per code symbol is $\frac{E_s}{N_0}$.

2.3 Assumptions

The assumptions for a system model may not be true in 'real' system. However, it is common to use some assumptions for a system which will help in the analysis of the system.

It is assumed that the source is *identically and independently distributed (i.i.d.)*. So, if the variables $\{U_k\}$ are independent, then,

$$\Pr(\boldsymbol{U}) = \prod_{k=1}^{K} \Pr(U_k) \tag{2.2}$$

If the variables $\{U_k\} \in \mathbb{F}_2$ are identical, then

$$\Pr(U_k) = \frac{1}{2} \qquad \forall \quad k = 1, 2 \ldots K \tag{2.3}$$

Combining equation (2.2) and (2.3) yields that all sequences are equiprobable.

$$\Pr(\boldsymbol{U}) = \frac{1}{2^K} \tag{2.4}$$

It is also assumed that the channel is a *memoryless channel without feedback*. Memoryless mean that the noise/cross over probability/erasure probability is independent for each symbol sent across the channel. Combining this with equation (2.2) yields.

$$\Pr(\,\boldsymbol{Y} \mid \boldsymbol{X}\,) = \prod_{n=1}^{N} \Pr(\,Y_n \mid X_n\,) \tag{2.5}$$

Moreover, a channel without feedback means that $\Pr(\,Y_n \mid \boldsymbol{X}\,) = \Pr(\,Y_n \mid X_n\,)$.

2.4 Log Likelihood Ratios

It is common to use ratio for a (binary) variable being one of two variables. Let us define the *log likelihood ratio* (LLR) or *L-value* as:

$$l = \lambda = L(\,y\,|\,x\,) = \ln\left(\frac{\Pr(\,y\,|\,x = 0\,)}{\Pr(\,y\,|\,x = 1\,)}\right) \tag{2.6}$$

The l-value describes how certain it is that x is 0 or 1. If l is positive then $\Pr(\,y\,|\,x = 0\,) > \Pr(\,y\,|\,x = 1\,)$, and the estimate should be $\hat{x} = 0$. If the value of $|l|$ is higher, then the reliability of the symbol will be higher. A hard decision rule of l is

$$\hat{x} = \begin{cases} 0 & l \geq 0 \\ 1 & l < 0 \end{cases} \tag{2.7}$$

For the different channels, the following l-values can be considered.

- BSC

$$L_{\text{BSC}}(\,y\,|\,x\,) = \begin{cases} \ln\frac{1-\epsilon}{\epsilon} & \text{for} \quad y = 0 \\ \ln\frac{\epsilon}{1-\epsilon} & \text{for} \quad y = 1 \end{cases} \tag{2.8}$$

- BEC

$$L_{\text{BEC}}(\,y\,|\,x\,) = \begin{cases} \infty & \text{for} \quad y = 0 \\ 0 & \text{for} \quad y = \Delta \\ -\infty & \text{for} \quad y = 1 \end{cases} \tag{2.9}$$

- BI-AWGNC

$$L_{\text{BI-AWGNC}}(\,y\,|\,x\,) = 4\frac{E_s}{N_0}y \tag{2.10}$$

2.5 Scaling of LLR

The scaling of LLR can be important for a channel considered during decoding.

Under BSC with ϵ the crossover probability (which also refers to the noise in the channel), the scaling of LLR (λ_n) by a constant β is possible as,

$$\beta \cdot \lambda_n = \beta \cdot \ln\left(\frac{\Pr(y_n|x_n = 0)}{\Pr(y_n|x_n = 1)}\right) = \begin{cases} \beta \cdot \ln\frac{1-\epsilon}{\epsilon} & \text{for } y_n = 0 \\ \beta \cdot \ln\frac{\epsilon}{1-\epsilon} & \text{for } y_n = 1 \end{cases} \tag{2.11}$$

So, we can always scale λ_n to ± 1 by β. Thus, it is not important to know the crossover probability (noise) in the BSC while using any decoding algorithm which allows scaling of LLR.

Similarly, under BEC the scaling of LLR is also possible as,

$$\beta \cdot \lambda_n = \beta \cdot \ln\left(\frac{\Pr(y_n|x_n = 0)}{\Pr(y_n|x_n = 1)}\right) = \begin{cases} \beta \cdot \infty & \text{for } y_n = 0 \\ \beta \cdot 0 & \text{for } y_n = \Delta \\ \beta \cdot -\infty & \text{for } y_n = 1 \end{cases} \tag{2.12}$$

This observation is also valid for AWGN channel. If the LLR (λ_n) for AWGN channel is multiplied by β,

$$\beta \cdot \lambda_n = \beta \cdot \left(4\frac{E_s}{N_0}y\right) \tag{2.13}$$

So, scaling signal to noise ratio $\frac{E_s}{N_0}$ by β implies that the knowledge of the noise is not necessary while determining λ_n for decoding.

2.6 Types of Decoding

When the codeword $\boldsymbol{x}$ is transmitted across a channel, the word received is $\boldsymbol{y}$ which is decoded to yield $\hat{\boldsymbol{x}}$. What could be the technique to decode $\boldsymbol{y}$ in order to estimate the transmitted codeword $\boldsymbol{x}$? One possible way to decode $\boldsymbol{y}$ is to maximize the a posteriori probability (MAP), and then guess the transmitted codeword.

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \; \Pr(\,\boldsymbol{x}\,|\,\boldsymbol{y}\,) \tag{2.14}$$

The decoding described in equation (2.14) is called *block-wise MAP decoding*. It is a block-wise because the entire block $\boldsymbol{y}$ is decoded to $\hat{\boldsymbol{x}}$.

If the maximization of a-posteriori probability (APP) is done symbol-wise rather than block-wise, then it will be called as *symbol-wise MAP*:

$$\hat{x}_n = \underset{x_n\in\mathcal{C}}{\operatorname{argmax}} \; \underbrace{\Pr(\,x_n\,|\,\boldsymbol{y}\,)}_{\text{APP}} \qquad n = 1, 2, \ldots, N \tag{2.15}$$

Where, $\hat{x}_n$ is an estimated code symbol and $\hat{x}_n \in \mathbb{F}_2$.

Now, if Bayes' rule is applied to (2.14).

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \; \frac{\Pr(\,\boldsymbol{x}\,,\,\boldsymbol{y}\,)}{\Pr(\,\boldsymbol{y}\,)} \tag{2.16}$$

$$= \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \; \Pr(\,\boldsymbol{y}\,|\,\boldsymbol{x}\,)\frac{\Pr(\,\boldsymbol{y}\,)}{\Pr(\,\boldsymbol{x}\,)} \tag{2.17}$$

If all the sequences are equiprobable as in equation 2.4, and the encoder is a one-to-one mapping, then $\Pr(\boldsymbol{x}) =$ constant. Since, maximization is only done over $\boldsymbol{x}$, the received word $\boldsymbol{y}$ can be considered as a constant.

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \Pr(\, \boldsymbol{y} \,|\, \boldsymbol{x}\,) \tag{2.18}$$

Equation (2.18) is the *block-wise maximum likelihood decoder (ML).*

Similarly, the *ML symbol-wise* can be derived from the MAP symbol-wise in the following way. So, from symbol-wise maximum a-posteriori probability (symbol-wise MAP),

$$\hat{x_n} = \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \quad \Pr(\, x_n \,|\, \boldsymbol{y}\,) \quad n = 1,\, 2,\, \dots\,,\, N \tag{2.19}$$

$$= \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \quad \frac{\Pr(x_n, \boldsymbol{y})}{\Pr(\boldsymbol{y})} \qquad \Big\{\text{Bayes' rule}\Big\} \tag{2.20}$$

$$= \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \quad \frac{\Pr(\, \boldsymbol{y} \,|\, x_n\,) \cdot \Pr(x_n)}{\Pr(\boldsymbol{y})} \tag{2.21}$$

Since, code symbols are equiprobable

$$= \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \quad \underbrace{\Pr(\, \boldsymbol{y} \,|\, x_n\,)}_{\text{ML symbol-wise}} \cdot \underbrace{\left(\frac{\Pr(x_n)}{\Pr(\boldsymbol{y})}\right)}_{\text{constant} = \alpha} \tag{2.22}$$

The ML block-wise and symbol-wise decoding are mostly pondered over in this report. The four possible decodings using MAP/ML and symbol/block-wise are shown in the Table 2.1.

Decoder type	Formula
ML block-wise	$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \Pr(\, \boldsymbol{y} \,|\, \boldsymbol{x}\,)$
ML symbol-wise	$\hat{x}_n = \underset{x_n\in\{0,1\}}{\operatorname{argmax}} \Pr(\, \boldsymbol{y} \,|\, x_n\,) \qquad \forall \quad n \,=\, 1\,,\, 2\,\dots N$
MAP block-wise	$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \Pr(\, \boldsymbol{x} \,|\, \boldsymbol{y}\,)$
MAP symbol-wise	$\hat{x}_n = \underset{x_n\in\{0,1\}}{\operatorname{argmax}} \Pr(\, x_n \,|\, \boldsymbol{y}\,) \qquad \forall \quad n \,=\, 1\,,\, 2\,\dots N$

Table 2.1: *Four different decoders.*

Binary Linear Block Codes 3

The goal of this chapter is to first give some definitions of important notions of *binary linear block codes*. Then, the representation of a factor graph and all its components will be described in section 3.2. Finally, section 3.3 will expound the low density parity check codes. Moreover, it can be accentuated that all the vectors used in this report are column vectors, but some equations are also shown considering row vectors. However, the distinction is made between the equations considering column vectors and row vectors.

3.1 Definitions

- A *(n,k) binary linear block code* is a finite set $\mathcal{C} \in \mathbb{F}_2^N$ of codewords $\boldsymbol{x}$. Each codeword is binary with length N. $\mathcal{C}$ contains 2^K codewords. The linearity of this code means that any linear combination of codewords is still a codeword [12].

  This is an example of a binary block code $(7,4)$:

$$\mathcal{C} = \begin{Bmatrix} (0000000), (1000101), (0100110), (0001111), \\ (0010011), (1100011), (1001010), (1101100), \\ (0110101), (1101100), (0011100), (1110000), \\ (1011001), (1011001), (0111010), (1111111). \end{Bmatrix} \tag{3.1}$$

- A code is generated by a *generator matrix* $\underline{\mathbf{G}} \in \mathbb{F}_2^{K\times N}$ and an *information word* $\boldsymbol{u} \in \mathbb{F}_2^K$ according to this formula:

$$\boldsymbol{x} = \boldsymbol{u} \otimes \underline{\mathbf{G}} \tag{3.2}$$

  where, $\boldsymbol{x}$ and $\boldsymbol{u}$ are row vectors.

  We work in $\mathbb{F}_2$, so the modulo 2 multiplication $\otimes$ is applied.

As only column vectors are used in the report, equation (3.2) becomes,

$$\boldsymbol{x} = \underline{\mathbf{G}}^T \otimes \boldsymbol{u} \tag{3.3}$$

where, $\boldsymbol{x}$ and $\boldsymbol{u}$ are column vectors. The set of all $\boldsymbol{u}$ represents all linear independent binary vectors of length K. For our example in the equation (3.1),

$$\boldsymbol{u} \in \begin{Bmatrix} (0000), (1000), (0100), (0010) \\ (0001), (1100), (1010), (1001) \\ (0110), (0101), (0011), (1110) \\ (1101), (1011), (0111), (1111) \end{Bmatrix} \tag{3.4}$$

The generator matrix $\underline{\mathbf{G}}$ corresponding to the code $\mathcal{C}$ is,

$$\underline{\mathbf{G}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.5}$$

Note that for a linear code, the corresponding rows of the generator matrix are the linear independent codewords.

- The *rate* of a code is given by the relation:

$$R = K/N \tag{3.6}$$

After the definition of the generator matrix, a definition of the dual matrix of this generator matrix could also be given.

- Indeed, $\underline{\mathbf{H}}$ is called the *parity check matrix* and is dual to $\underline{\mathbf{G}}$ because:

$$\underline{\mathbf{H}} \otimes \underline{\mathbf{G}}^T = \mathbf{0} \tag{3.7}$$

$\underline{\mathbf{H}}$ belongs to $\mathbb{F}_2^{M \times N}$ and satisfies:

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = \mathbf{0}_{M \times 1} \qquad \forall \;\; \boldsymbol{x} \in \mathcal{C} \tag{3.8}$$

The rank of $\underline{\mathbf{H}}$ is $N - K$ and often, $M \geq N - K$.

The matrix $\underline{\mathbf{H}}$ of our example in the equation (3.1) is:

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.9}$$

- Finally, let us define the *minimum distance* d_{min}. The minimum distance for a linear code is the minimum hamming weight of all nonzero codewords of the code, with the hamming weight of a codeword corresponding to the number of $1s$ in this codeword [8].

$$d_{min} = \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{C} \backslash \mathbf{0}} w_H(\boldsymbol{x}) \tag{3.10}$$

For example, the hamming weight of (1000101) is 3.

Thus, always in our example in the equation (3.1), the minimum distance found is 3.

The code $\mathcal{C}$ of equation (3.1) is a $(7, 4, 3)$ linear block code. This code, called a *Hamming code*, will often be used as an example in the report and in the following section 3.2 particularly.

3.2 Factor Graphs

A *factor graph* is the representation with edges and nodes of the parity check equation:

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = \mathbf{0}_{M \times 1} \tag{3.11}$$

Each *variable node* corresponds to a code symbol of the codeword and each *check node* represents one check equation. An *edge* links a variable node to a check node. It corresponds to a 1 in the parity check matrix $\underline{\mathbf{H}}$. A factor graph is a bipartite graph, which means that there are two kind of nodes and the same kind of nodes are never connected directly with an edge.

If the example of the $(7, 4, 3)$ Hamming code is taken again, and it is considered that x_n is a code symbol of the codeword $\boldsymbol{x}$, the set of parity check equation is,

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = \left\{ \begin{matrix} chk(A) : x_1 \oplus x_2 \oplus x_4 \oplus x_5 = 0 \\ chk(B) : x_2 \oplus x_3 \oplus x_4 \oplus x_6 = 0 \\ chk(C) : x_4 \oplus x_5 \oplus x_6 \oplus x_7 = 0 \end{matrix} \right\} \tag{3.12}$$

Figure 3.1 shows two factor graphs of a $(7, 4, 3)$ Hamming code. These factor graphs have 3 check nodes,7 variables nodes and, for example, there is an edge between each variables x_1, x_2, x_4 and x_5 and the check node A.

As it can be seen in this Figure 3.1, there are some *cycles* in this graph, for instance the cycle formed by x_2, A, x_4 and B. The presence of the cycles in a factor graph can sometimes make the decoding difficult, as it will be shown later in the report. And of course, graphs without cycles also exist and can be represented as a tree graph. The

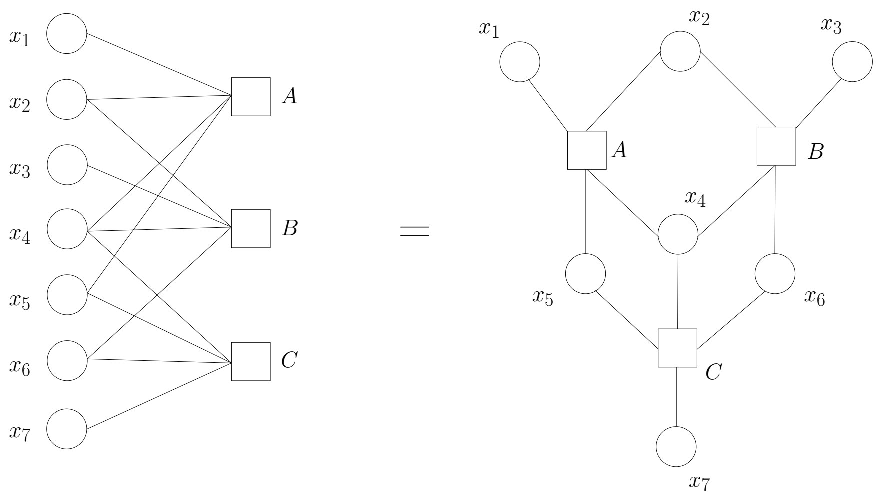


Figure 3.1: *Factor graphs of a* $(7, 4, 3)$ *Hamming code.*

Figure 3.2 corresponding to the following parity check matrix is an example of a factor graph without cycle.

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.13}$$

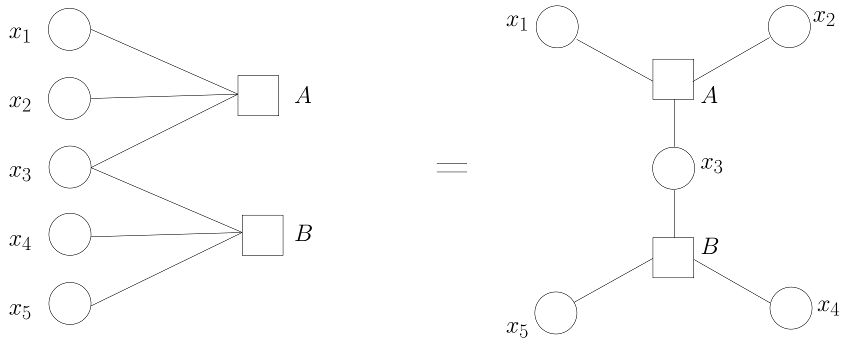


Figure 3.2: *A factor graph without cycles.*

While using the factor graph, it is frequently useful to have the set of neighbours of both variable and check nodes. So, let us define what these sets of neighbours are.

- The set of indices of variables nodes, which are in the neighbourhood of a check node m, is called $\mathcal{N}(m)$. For instance $(7, 4, 3)$ Hamming code in the Figure 3.1,

$$\mathcal{N}(B) = \{2, 3, 4, 6\}$$

- In the same way, $\mathcal{M}(n)$ is defined as the set of check nodes linked to the variable node n. According to Figure 3.1,

$$\mathcal{M}(4) = \{A, B, C\}$$

Factor graphs are a nice graphical way to represent the environment of decoding and is also used for the message passing algorithm which will be described in chapter 4.

3.3 Low Density Parity Check Codes

In the following section, the Low Density Parity Check (LDPC) Codes will be introduced and described.

LDPC codes were discovered by Robert Gallager in the $60s$. They were forgotten for almost 30 years before being rediscovered again thanks to their most important advantage, which is that they allow data transmission rates close to the Shannon limit, the theoretical rate [21]. A design of a LDPC code, which comes within $0.0045\ dB$ of the Shannon limit, has been found [4]. This discovery motivates the interest of researchers on LDPC codes and decoders, as the decoding gives a really small probability of lost information. LDPC codes are becoming now the standard in error correction for applications such as mobile phones and satellite transmission of digital television [18][21].

LDPC codes can be classified in two categories, *regular* and *irregular* LDPC codes.

- A regular LDPC code is characterized by two values: d_v, and d_c.
  d_v is the number of ones in each column of the parity check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M \times N}$.
  d_c represents the number of ones in each row.

- There are two different rates in LDPC codes.

  The *true rate* is the normal rate:

$$R = K/N \tag{3.14}$$

  The second rate is called as *design rate*:

$$R_d = 1 - \frac{d_v}{d_c} \tag{3.15}$$

  The relation between those two rates is:

$$R \geq R_d \tag{3.16}$$

Let us prove the equation(3.16):

The number of ones in the parity check matrix $\underline{\mathbf{H}}$ is: $Md_c = Nd_v$. In $\underline{\mathbf{H}}$, some check equations can be repeated; so the number of rows M could be greater or equal to (N-K). Thus,

$$R = \frac{K}{N} = \frac{N-(N-K)}{N} = 1 - \frac{N-K}{N} \geq 1 - \frac{M}{N} = 1 - \frac{d_v}{d_c}$$

The following equation represents a regular LDPC code with its parameters:

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.17}$$

$$d_v = 2;\ d_c = 3;\ M = 4;\ N = 6;\ rank(\underline{\mathbf{H}}) = 4$$

$$K = N - rank(\underline{\mathbf{H}}) = 2;\ R = R_d = 1/3.$$

Note that a real LDPC code, as its name tells it, has a small number of ones in rows and columns compare to its really large dimensions. LDPC codes can work well for code length $N > 1000$ [11], so for instance the dimensions of the code can be $N = 1000,\ M = 500$ and the degrees $d_v = 5, d_c = 10$.

- An irregular LDPC code is a code with different numbers of ones in each row and columns. They are known to be better than the regular one [10]. According to this difference, new variables are defined for these irregular LDPC codes (see appendix A).

  The following example is not a real irregular LDPC code but it is an irregular linear code and will help to understand what can be an irregular LDPC code.

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.18}$$

We can see in the Figure 3.3 that the number of ones in some columns is 3 and in others it is 2. We have also the same situation for rows, some rows have 3 ones and others have 2 ones.

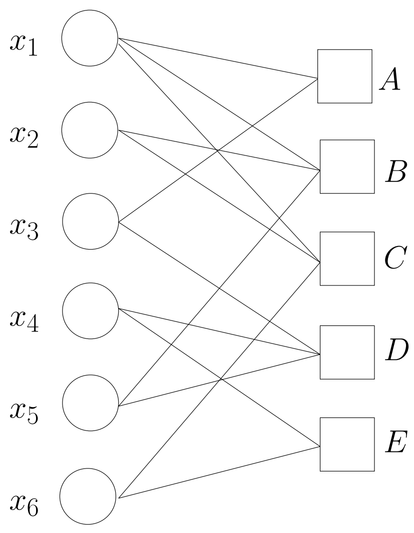


Figure 3.3: *Irregular LDPC Code.*

4 Message Passing Algorithm

Linear block and LDPC codes can iteratively be decoded on the factor graph by *Message-Passing Algorithm* (MPA) (it is also known as *Sum-Product (Belief/Probability Propagation) or Max-Product (Min-Sum) Algorithm* [11]). As it is known that a factor graph (cf. section 3.2) represents a factorization of the *global code constraint*

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = \mathbf{0}$$

into the *local code constraints* which are represented by the connection between variable and check nodes. These nodes perform local decoding operations and exchange the *messages* along the edges of the factor graph. It can be construed that the extrinsic message is a soft-value for a symbol when the direct observation of the symbol is not considered in the computation (local decoding operation) of this specific value.

The message passing algorithm is an iterative decoding technique. So, in the first iteration, the incoming messages received from the channel at the variable nodes are directly passed along the edges to the neighbouring check nodes because there are no incoming messages (extrinsic) from the check nodes in the first iteration. The check nodes perform local decoding operations to compute outgoing messages (extrinsic) depending on the incoming messages received from the neighbouring variable nodes. Thereafter, these new outgoing messages are sent back along the edges to the neighbouring variable nodes. The meaning of one complete iteration can be comprehended that the one outgoing message (extrinsic) has passed in both directions along every edge. One iteration is illustrated in the Figure 4.1 for the $(7, 4, 3)$ Hamming code by showing the direction of the message in each direction along every edge. The variable-to-check (μ_{vc}) and check-to-variable (μ_{cv}) are extrinsic messages which are also shown in the same Figure 4.1.

After every one complete iteration, it will be checked whether a valid codeword is

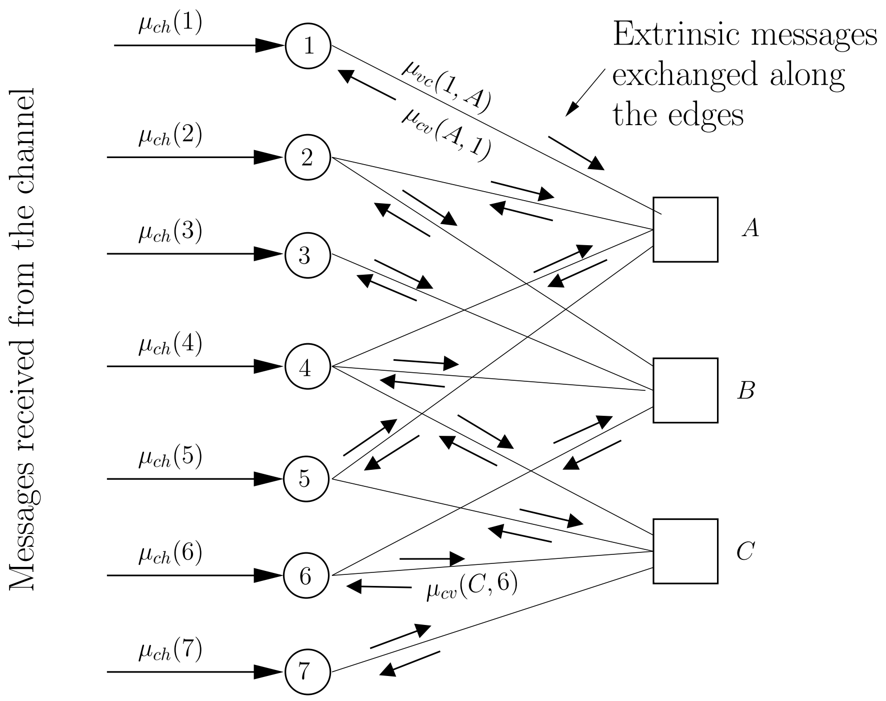


Figure 4.1: *To illustrate one complete iteration in a factor graph for* $(7,4,3)$ *Hamming code. The messages for instance* $\mu_{vc}(1,A)$ *and* $\mu_{cv}(A,1)$ *are extrinsic.* μ_{ch} *are the messages coming from the channel.*

found or not. If the estimated code symbols form a valid codeword such that

$$\underline{\mathbf{H}} \otimes \hat{\boldsymbol{x}} = \mathbf{0}$$

(where, $\hat{\boldsymbol{x}}$ is an estimated codeword.)

then the iteration will be terminated otherwise it will continue. After the first complete iteration, the variable nodes will perform the local decoding operations in the same way to compute the outgoing messages (extrinsic) from the incoming messages received from both the channel and the neighbouring check nodes. In this way, the iterations will continue to update the extrinsic messages unless the valid codeword is found or some stopping criterion is fulfilled.

4.1 Message Passing Algorithm and Node Operations

Considering any factor graph in general, the message passing algorithm is listed below in order to give an overview of this algorithm. The extrinsic messages which are computed by the local decoding operations at the variable nodes are denoted as μ_{vc} which means message from variable $\rightarrow$ check while at the check nodes are denoted as μ_{cv} which means message from check $\rightarrow$ variable.

1. The initial message coming from the channel at variable node n is denoted as $\mu_{ch}(n)$.
2. The extrinsic message from the variable to check node is
$$\mu_{vc}(n,m) = fct_v\big(\mu_{ch}(n),\ \mu_{cv}(m',n)\big) \tag{4.1}$$
where,
   - $n =$ variable node
   - $m \in \mathcal{M}(n)$: check nodes which are the neighbour of the variable node n.
   - $m' \in \mathcal{M}(n)\backslash m$: check nodes except m which are the neighbour of the variable node n.

   The new or updated extrinsic message $\mu_{vc}(n,m)$ which is computed by the local decoding operation or function fct_v, will be sent to the check node m. Therefore, the incoming extrinsic message $\mu_{cv}(m,n)$ from the check node m is not considered for updating the message $\mu_{vc}(n,m)$.
3. The extrinsic message from the check to variable node is
$$\mu_{cv}(m,n) = fct_c\big(\mu_{vc}(n',m)\big) \tag{4.2}$$
where, fct_c is the local decoding operation at a check node and $n' \in \mathcal{N}(m)\backslash n$: variable nodes except n which are the neighbour of the check node m.
4. The final message that is computed at the variable node n in order to estimate the code symbol.
$$\mu_v(n) = fct_v\big(\mu_{ch}(n),\ \mu_{cv}(m,n)\big) \tag{4.3}$$
5. The estimation of a code symbol X_n can be done by hard decision
$$\hat{x}_n = \begin{cases} 0 & \text{if } \Pr(\,X_n = 0 \,|\, \mu_v(n)\,) \geq \Pr(\,X_n = 1 \,|\, \mu_v(n)\,) \\ 1 & \text{else} \end{cases} \tag{4.4}$$
6. If these symbol-wise estimated code symbols are stacked to form vector $\hat{\boldsymbol{x}}$ of length N, then it can be checked whether the $\hat{\boldsymbol{x}}$ is a valid codeword by
$$\underline{\mathbf{H}} \otimes \hat{\boldsymbol{x}} = \mathbf{0} \tag{4.5}$$
7. If the above equation (4.5) is satisfied or the current number of iteration is equal to some defined maximum number of iterations then stop the iteration otherwise repeat the algorithm from step 2 to step 7.

4.1.1 Example of Node Operations

Considering the factor graph of $(7, 4, 3)$ Hamming code which is shown in Figure 4.1, the above steps the algorithm is shown.

of algorithm can be seen.

At the variable node 2:

- The initial message at the variable node 2 is $\mu_{ch}(2)$.
- In general, the incoming (extrinsic) messages at the node 2:

$$\mu_{ch}(2),\ \mu_{cv}(A,2),\ \mu_{cv}(B,2) \tag{4.6}$$

- In general, the outgoing (extrinsic) message from the node 2:

$$\mu_{vc}(2,A),\ \mu_{vc}(2,B) \tag{4.7}$$

- The local decoding operation at the node 2 to compute the extrinsic(outgoing) message say $\mu_{vc}(2,B)$:

$$\mu_{vc}(2,B) = fct_v\big(\mu_{ch}(2),\ \mu_{cv}(A,2)\big) \tag{4.8}$$

  So, the message $\mu_{cv}(B,2)$ is excluded for the computation of $\mu_{vc}(2,B)$.

At the check node B:

- The incoming (extrinsic) messages at the check node B:

$$\mu_{vc}(2,B),\ \mu_{vc}(3,B),\ \mu_{vc}(4,B),\ \mu_{vc}(6,B) \tag{4.9}$$

- The outgoing (extrinsic) message from the check node B:

$$\mu_{cv}(B,2),\ \mu_{cv}(B,3),\ \mu_{cv}(B,4),\ \mu_{cv}(B,6) \tag{4.10}$$

- the local decoding operation at the check node B to compute the extrinsic (outgoing) message say $\mu_{cv}(B,2)$:

$$\mu_{cv}(B,2) = fct_c\big(\mu_{vc}(3,B),\ \mu_{vc}(4,B),\ \mu_{vc}(6,B)\big) \tag{4.11}$$

  It can be noticed that the message $\mu_{vc}(2,B)$ is excluded for the computation of $\mu_{cv}(B,2)$.

It shall be noted that these messages μ can be in terms of either probabilities or log likelihood ratios LLR.

4.2 Definitions and Notations

Some terms and notations are introduced here to show extrinsic, a-posteriori and intrinsic probabilities only for the simplification of further derivations and proofs. We assume that these short notations won't be repellent to the reader. However, these notations are easy to assimilate as this report goes on.

If codeword $\boldsymbol{x}$ having length N is sent across any memoryless channel and $\boldsymbol{y}$ is a received word such that the *extrinsic a-posteriori probability* after decoding is given in equation (4.12) for the n^{th} symbol to be $b = \{0, 1\}$

$$p^b_{e,n} = \Pr(\, X_n = b \,|\, \boldsymbol{y}_{\setminus n}\,) \qquad n = 1, 2, \ldots, N;\; b = 0, 1 \tag{4.12}$$

where, $\boldsymbol{y}_{\setminus n} = [y_1\ y_2\ \ldots\ y_{n-1}\ y_{n+1}\ y_N]^T$

$\boldsymbol{y}_{\setminus n}$ means y_n is excluded from the received word $\boldsymbol{y}$

However, the *a-posteriori probability* (APP) after decoding is given in equation (4.13) to show the disparity between APP and extrinsic a-posteriori probability,

$$p^b_{p,n} = \Pr(\, X_n = b \,|\, \boldsymbol{y}\,) \qquad n = 1, 2, \ldots, N;\; b = 0, 1 \tag{4.13}$$

So, the difference between APP and extrinsic a-posteriori probability can easily be seen that only y_n is excluded from the received word $\boldsymbol{y}$ in the formulation of extrinsic a-posteriori probability. Now, one more definition thrusts here to introduce *intrinsic probability* before decoding which is defined as

$$p^b_{ch,n} = \Pr(\, y_n \,|\, X_n = b\,) \qquad n = 1, 2, \ldots, N;\; b = 0, 1 \tag{4.14}$$

The channel is assumed to be *binary-input memoryless symmetric channel*. The channel properties are reiterated here to prove the *independency assumption* in the factor graph. So, the channel being *binary-input* means that the data transmitted is a discrete symbol from Galois Field $\mathbb{F}_2$ i.e., $\{0, 1\}$, *memoryless* means that each symbol is affected independently by the noise in the channel and *symmetric* means the noise in the channel affects the $0s$ and $1s$ in the same way. As, there is no direct connection between any two variable nodes in the factor graph, the decoding of the code symbol can be pondered on each variable node independently. It means that the local decoding operations can be performed independently at both variable and check nodes side in the factor graph. If the factor graph is cycle free, then the independency assumption is valid whereas if the factor graph has *cycles*, then the assumption will be valid for few iterations until the messages have travelled the entire cycles.

The MPA algorithm is *optimal* $\big($i.e., maximum likelihood (ML) decoding$\big)$ for those codes whose factor graph is cycle free otherwise *sub-optimal* due to cycles in the factor graph. So, if the codes have cycles then still the *MPA decoder will perform close to ML decoder* [9]. Furthermore, the overall decoding complexity is linear with the code length [15]. So, these are the motivations behind studying the MPA decoder.

In this chapter, sum-product and max-product/min-sum algorithms are described and the performance of the message passing algorithm is also explained under binary-erasure channel (BEC). The BEC is considered because it is easy to explain the concepts behind the update rules. The *stopping set* (a set of code symbols which is not resolvable) is also explained in detail under BEC.

4.3 Sum-Product Algorithm

The sum-product algorithm was invented by Gallager [6] as a decoding algorithm for LDPC codes which is still the standard algorithm for the decoding of LDPC codes [11]. The sum-product algorithm operates in a factor graph and attempts to compute various *marginal functions* associated with the *global function or global code constraint* by iterative computation of local functions or local code constraints.

In this section, the sum-product algorithm is shown as a method for maximum likelihood symbol-wise decoding. The update rules for independent and isolated variable and check nodes in terms of probabilities and LLR (L values) are derived. The algorithm is explained in an intuitive way such that it can show the concepts behind it.

4.3.1 Maximum Likelihood

The property of the sum-product algorithm is that for cycle free factor graphs it performs maximum likelihood (ML) symbol-wise decoding. In this section, the sum-product formulation is derived from ML symbol-wise (cf. section 2.6 for types of decoding).

Considering the linear block code $\mathcal{C} \in \mathbb{F}_2^N$, where $|\mathcal{C}| = 2^K$, information word length K, code rate $R = \frac{K}{N}$. The code is defined by a parity check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M \times N}$, $M \geq N - K$.

Formally,

$$\begin{aligned}\mathcal{C} &= \Big\{\boldsymbol{x} : \boldsymbol{x} = enc(\boldsymbol{u}),\ \text{information word } \boldsymbol{u} \in \mathbb{F}_2^K\Big\} \\ &= \Big\{\boldsymbol{x} \in \mathbb{F}_2^N\ :\ \underline{\mathbf{H}}\ \otimes\ \boldsymbol{x} = \mathbf{0}\Big\}\end{aligned}$$

If the code $\mathcal{C}$ has cycle free factor graph and a codeword $\boldsymbol{x} \in \mathcal{C}$ is transmitted through any binary-input memoryless channel then the sum-product algorithm can decode or estimate an optimal (ML) codeword from the received word $\boldsymbol{y}$. It is assumed that the code symbols and codewords are equiprobable.

In general, if the code symbols are equiprobable then maximum a-posteriori probability (MAP) $\Pr(\, x_n \,|\, \boldsymbol{y} \,)$ and maximum likelihood (ML) $\Pr(\, \boldsymbol{y} \,|\, x_n \,)$ are same.

Let's contemplate that $\hat{x}_i$ is an estimated code symbol, such that $\hat{x}_i \in \mathbb{F}_2$.

So, symbol-wise ML:

$$\hat{x_n} = \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \Pr(\, \boldsymbol{y} \,|\, x_n \,) \qquad n = 1, 2, \ldots, N \tag{4.15}$$

$$= \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \frac{\Pr(x_n, \boldsymbol{y})}{\underbrace{\Pr(x_n)}_{\text{constant}}} \qquad \left\{\begin{array}{c}\text{Applying Bayes' rule;}\\ \text{code symbols } x_n \text{ are equiprobable}\end{array}\right\} \tag{4.16}$$

$$= \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \Pr(x_n, \boldsymbol{y}) \tag{4.17}$$

$$= \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \Pr(\boldsymbol{x}, \boldsymbol{y}) \tag{4.18}$$

$$= \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \Pr(\, \boldsymbol{y} \,|\, \boldsymbol{x} \,)\, \underbrace{\Pr(\boldsymbol{x})}_{\text{constant}} \qquad \left\{\begin{array}{c}\text{since, codewords}\\ \boldsymbol{x} \text{ are equiprobable}\end{array}\right\} \tag{4.19}$$

$$= \underset{x_n \in \{0,1\}}{\text{argmax}} \quad \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \Pr(\, \boldsymbol{y} \,|\, \boldsymbol{x} \,)$$

Since, channel is assumed to be memoryless

$$\hat{x_n} = \underbrace{\underset{x_n \in \{0,1\}}{\text{argmax}}}_{Decision} \underbrace{\sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}}}_{Sum} \underbrace{\prod_{j=1}^{N}}_{Product} \Pr(\, y_j \,|\, x_j \,) \tag{4.20}$$

So, the sum-product formulation can be derived from the maximum likelihood symbol-wise for any code. However, if there are cycles in the factor graph of the code, then the factor graph will not have tree structure and the sum-product algorithm will be sub-optimal but close to the ML decoder [10].

4.3.2 The General Formulation of APP

The general form of APP can easily be construed from equation (4.20) and (2.22) i.e.,

$$\text{(APP)} \quad p_{p,n}^{x_n} = \Pr(\, X_n = x_n \,|\, \boldsymbol{y} \,) \tag{4.21}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{j=1}^{N} \Pr(\, y_j \,|\, X_j = x_j \,) \qquad \Big\{\text{where, } \alpha \text{ is scaling factor}\Big\} \tag{4.22}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{j=1}^{N} p_{ch,j}^{x_j} \tag{4.23}$$

So, the above equation can also be written as

$$p_{p,n}^{0} = \Pr(\, X_n = 0 \,|\, \boldsymbol{y} \,) \;=\; \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n = 0}} \prod_{j=1}^{N} p_{ch,j}^{x_j} \tag{4.24}$$

$$p_{p,n}^{1} = \Pr(\, X_n = 1 \,|\, \boldsymbol{y} \,) \;=\; \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n = 1}} \prod_{j=1}^{N} p_{ch,j}^{x_j} \tag{4.25}$$

with scaling factor α such that $p_{p,n}^{0} + p_{p,n}^{1} = 1$.

4.3.3 The General Formulation of Extrinsic A-Posteriori Probability

The general formulation of extrinsic a-posteriori probability can be derived from the general form of APP using equation (4.22).

$$\text{(APP)} \quad p_{p,n}^{x_n} = \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \Pr(\, y_n \,|\, X_n = x_n \,) \cdot \left\{ \prod_{\substack{j=1 \\ j \neq n}}^{N} p_{ch,j}^{x_j} \right\} \tag{4.26}$$

$$= \Pr(\, y_n \,|\, X_n = x_n \,) \cdot \left\{ \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{\substack{j=1 \\ j \neq n}}^{N} p_{ch,j}^{x_j} \right\} \tag{4.27}$$

$$= \underbrace{\Pr(\, y_n \,|\, X_n = x_n \,)}_{\substack{p_{ch,n}^{x_n} \\ \text{Intrinsic Probability}}} \cdot \underbrace{\left\{ \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{\substack{j=1 \\ j \neq n}}^{N} \Pr(\, y_j \,|\, X_j = x_j \,) \right\}}_{\substack{\Pr(\, X_n = x_n \,|\, \boldsymbol{y}_{\setminus n} \,) \\ \text{Extrinsic Probability}}} \tag{4.28}$$

So, Extrinsic a-posteriori probability in general is

$$p_{e,n}^{x_n} = \Pr(\, X_n = x_n \,|\, \boldsymbol{y}_{\setminus n} \,) \tag{4.29}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{\substack{j=1 \\ j \neq n}}^{N} \Pr(\, y_j \,|\, X_j = x_j \,) \tag{4.30}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n \text{ fixed}}} \prod_{\substack{j=1 \\ j \neq n}}^{N} p_{ch,j}^{x_j} \tag{4.31}$$

Moreover, the extrinsic a-posteriori probability can be rewritten as

$$p_{e,n}^{0} = \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n = 0}} \prod_{\substack{j=1 \\ j \neq n}}^{N} p_{ch,j}^{x_j} \tag{4.32}$$

$$p_{e,n}^{1} = \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_n = 1}} \prod_{\substack{j=1 \\ j \neq n}}^{N} p_{ch,j}^{x_j} \tag{4.33}$$

with scaling factor α such that $p_{e,n}^{0} + p_{e,n}^{1} = 1$.

4.3.4 Intrinsic, A-posteriori and Extrinsic L-values

Before getting further, it can be accentuated that log likelihood ratios (LLR or L-values) of the intrinsic, a-posteriori and extrinsic can be shown now as log ratios of the intrinsic, a-posteriori and extrinsic probabilities formulations respectively. In fact, *L-values have lower complexity and are more convenient to use than the messages in terms of probabilities*. Moreover, the *log likelihood ratio is a special L-value* [9].

Intrinsic L-value:

$$l_{ch,n} = L(\, y_n \,|\, X_n \,) = \ln\left(\frac{\Pr(\, y_n \,|\, X_n = 0\,)}{\Pr(\, y_n \,|\, X_n = 1\,)}\right) = \ln\left(\frac{p_{ch,n}^{0}}{p_{ch,n}^{1}}\right) \tag{4.34}$$

A-posteriori L-value:

$$l_{p,n} = L(\, X_n \,|\, \boldsymbol{y} \,) = \ln\left(\frac{\Pr(\, X_n = 0 \,|\, \boldsymbol{y}\,)}{\Pr(\, X_n = 1 \,|\, \boldsymbol{y}\,)}\right) = \ln\left(\frac{p_{p,n}^{0}}{p_{p,n}^{1}}\right) \tag{4.35}$$

Extrinsic L-value:

$$l_{e,n} = L(\, X_n \,|\, \boldsymbol{y}_{\setminus n} \,) = \ln\left(\frac{\Pr(\, X_n = 0 \,|\, \boldsymbol{y}_{\setminus n}\,)}{\Pr(\, X_n = 1 \,|\, \boldsymbol{y}_{\setminus n}\,)}\right) = \ln\left(\frac{p_{e,n}^{0}}{p_{e,n}^{1}}\right) \tag{4.36}$$

It should be noted that the intrinsic, a-posteriori and extrinsic probabilities can also be found, if the respective L-values (LLR) are given. For the convenience, all the notations in the subscripts can be removed from both L-values (as, l) and probabilities (as, p) to derive the relation i.e., $\Pr(\, X \,|\, l\,)$.

Since,

$$p^0 + p^1 = 1 \tag{4.37}$$

and

$$l = \ln\left(\frac{p^0}{p^1}\right) \tag{4.38}$$

So, from the above two relations (4.37) and (4.38),

$$l = \ln \frac{p^0}{1-p^0} \tag{4.39}$$

$$\Leftrightarrow \left(1-p^0\right) \cdot e^l = p^0 \tag{4.40}$$

$$\Leftrightarrow e^l = \left(1+e^l\right) \cdot p^0 \tag{4.41}$$

$$\Leftrightarrow p^0 = \frac{e^{-l/2}}{e^{-l/2}} \cdot \frac{e^l}{\left(1+e^l\right)} \tag{4.42}$$

$$\Leftrightarrow p^0 = \Pr(\, X = 0 \,|\, l \,) = \frac{e^{+l/2}}{\left(e^{-l/2}+e^{+l/2}\right)} \tag{4.43}$$

Similarly, it can be found for

$$p^1 = \Pr(\, X = 1 \,|\, l \,) = \frac{e^{-l/2}}{\left(e^{-l/2}+e^{+l/2}\right)} \tag{4.44}$$

As it is known that the sum-product algorithm does local decoding operations at both variable nodes and check nodes individually and independently in order to update the extrinsic messages iteratively unless the valid codeword is found or some other stopping criterion is fulfilled. In the next section, the update rules are shown for both variable and check nodes which are the sum-product formulation basically.

4.3.5 Sum-Product Message Update Rules

The property of the message passing algorithm is that the individual and independent variable nodes have *repetition code constraints* while the check nodes have *single parity check code constraints* which is proved in this subsection. The Forney factor graphs (FFG) [11] are considered to prove the repetition code constraints at the variable nodes and the single parity check code constraints at the check nodes because of simplicity. It shall be accentuated that the Forney style factor graphs and the factor graphs/Tanner graph/Tanner-Wiberg graph [9] have same code constraints at variable and check nodes, so they are same but with different representations. For more information regarding Forney style factor graphs refer [11]. In this subsection, the the update rules are also shown in terms of probabilities and log likelihood ratios (L-values) in an intuitive way considering small examples before it is generalized.

4.3.5.1 Message Update Rules for Variable Nodes

The standard coding model is considered in which a codeword $\boldsymbol{x}$ having length N is selected from a code $\mathcal{C} \in \mathbb{F}_2^N$, $|\mathcal{C}| = 2^K$ and transmitted across a memoryless channel with the corresponding received word $\boldsymbol{y}$ having length N.

A small example is considered whose code is represented by a parity-check matrix

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.45}$$

and the Forney style factor graph with the coding model is shown in Figure 4.2 and the individual variable node 3 of the Figure 4.2 is shown in Figure 4.3. The repetition code constraint for a variable node is proved in an intuitive way.

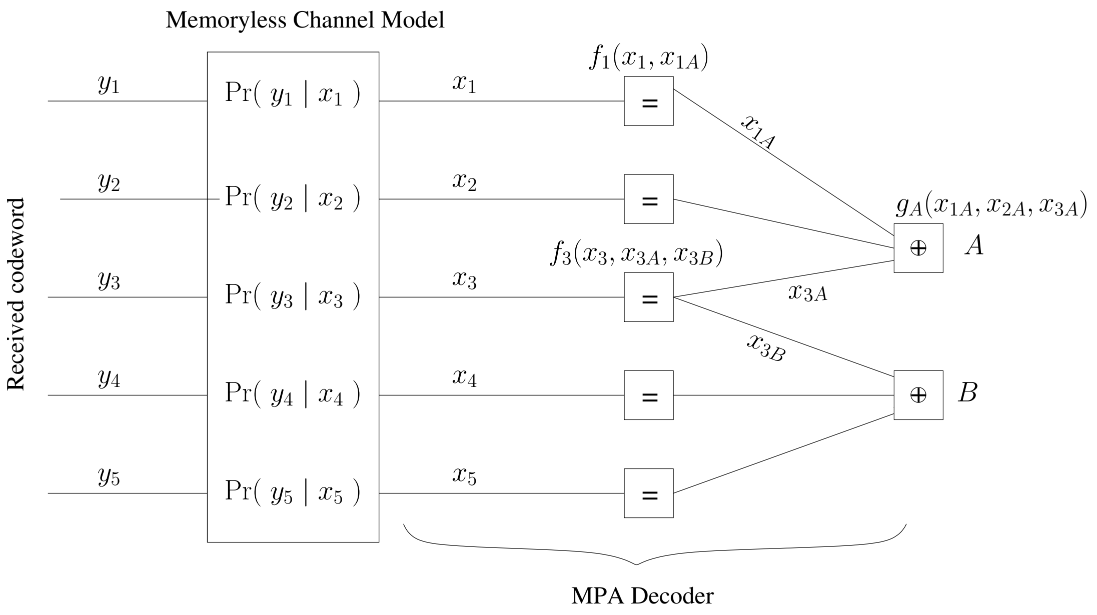


Figure 4.2: *Forney style factor graph (FFG) with the coding model of the code defined by the check matrix in equation* (4.45).

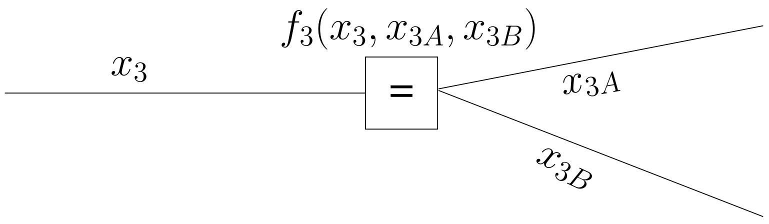


Figure 4.3: *The individual and isolated variable node* 3 *of the Figure 4.2.*

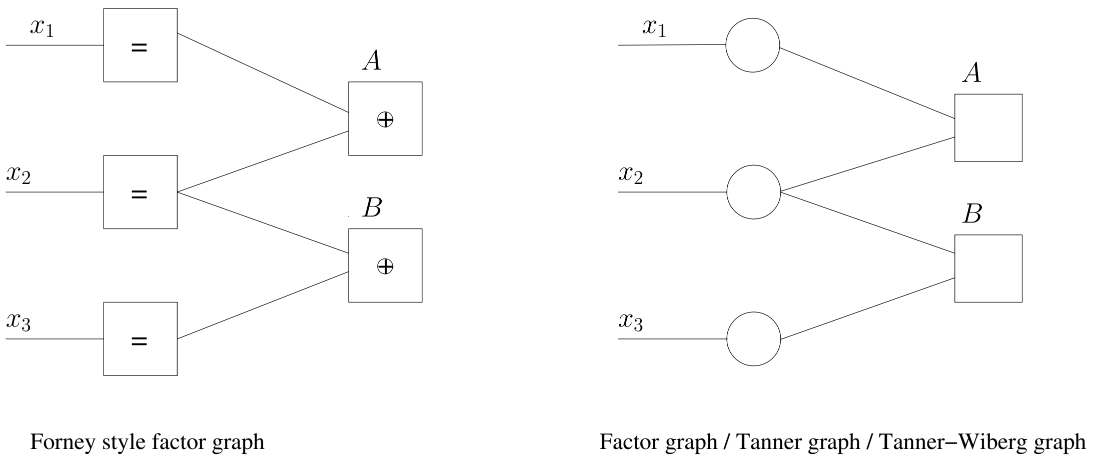
The indicator function of a variable node 3 is defined as $f_3(x_3,\ x_{3A},\ x_{3B})$ [11] where x_3, x_{3A} and x_{3B} are the variables, such that

$$f_3(x_3,\ x_{3A},\ x_{3B}) = \begin{cases} 1, & \text{if } x_3 = x_{3A} = x_{3B} \\ 0, & \text{otherwise} \end{cases} \tag{4.46}$$

So, the equation (4.46) implies that all the edges which are connected to a variable node have got the same value in the variables i.e., $x_3 = x_{3A} = x_{3B} = 0$ or $x_3 = x_{3A} = x_{3B} = 1$ like a repetition code. Thus, the message update rules of the variable nodes can be defined considering the variable node has got the repetition code constraint.

Now a small example of repetition code having length $N = 3$ is considered to explain the message update rules for variable nodes having degree 3. Thereafter, the message update rules for variable nodes are generalized. It is also shown that these message update rules are instances of sum-product formulation. Let the repetition/repeat code be $\mathcal{C} = \{000, 111\}$ such that the codeword $\boldsymbol{x} = [x_1\ x_2\ x_3]^T \in \mathcal{C}$ is transmitted across the memoryless channel and the received word is $\boldsymbol{y} = [y_1\ y_2\ y_3]^T$. The FFG and Factor graph of the repetition code of length $N = 3$ is shown in Figure 4.4 which is represented by the parity-check matrix,

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.47}$$



x_1
x_2
x_3
=
=
=
A
B
⊕
⊕
Forney style factor graph
x_1
x_2
x_3
A
B
Factor graph / Tanner graph / Tanner–Wiberg graph

Figure 4.4: *Forney factor graph (FFG) and the factor graph of the repetition code of length $N = 3$.*

The extrinsic message of the code symbol x_2 is considered to explain the update rules for the repetition code of length $N = 3$. So, from equation (4.32) and (4.33),

$$\Pr(\, x_2 = 0 \,|\, \boldsymbol{y}_{\backslash 2}\,) = \Pr(\, x_2 = 0 \,|\, y_1,\ y_3\,) = \underbrace{p^0_{e,2}}_{\mu_2(0)} \tag{4.48}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_2 = \underline{0}}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j\,) \tag{4.49}$$

$$= \alpha \cdot \sum_{\boldsymbol{x} = [0\underline{0}0]} \Pr(\, y_1 \,|\, X_1 = 0\,) \cdot \Pr(\, y_3 \,|\, X_3 = 0\,) \tag{4.50}$$

$$= \alpha \cdot \Pr(\, y_1 \,|\, X_1 = 0\,) \cdot \Pr(\, y_3 \,|\, X_3 = 0\,) \tag{4.51}$$

$$p^0_{e,2} = \alpha \cdot \underbrace{p^0_{ch,1}}_{\mu_1(0)} \cdot \underbrace{p^0_{ch,3}}_{\mu_3(0)} \tag{4.52}$$

Similarly,

$$\Pr(\, x_2 = 1 \,|\, \boldsymbol{y}_{\backslash 2}\,) = \Pr(\, x_2 = 1 \,|\, y_1,\ y_3\,) = \underbrace{p^1_{e,2}}_{\mu_2(1)} \tag{4.53}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_2 = \underline{1}}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j\,) \tag{4.54}$$

$$= \alpha \cdot \sum_{\boldsymbol{x} = [1\underline{1}1]} \Pr(\, y_1 \,|\, X_1 = 1\,) \cdot \Pr(\, y_3 \,|\, X_3 = 1\,) \tag{4.55}$$

$$= \alpha \cdot \Pr(\, y_1 \,|\, X_1 = 1\,) \cdot \Pr(\, y_3 \,|\, X_3 = 1\,) \tag{4.56}$$

$$p^1_{e,2} = \alpha \cdot \underbrace{p^1_{ch,1}}_{\mu_1(1)} \cdot \underbrace{p^1_{ch,3}}_{\mu_3(1)} \tag{4.57}$$

The scaling factor α can be found such that $p^0_{e,2} + p^1_{e,2} = 1$.

In terms of log likelihood ratios (L-values)

The message update rule for the variable node 2 in terms of log likelihood ratios are,

$$\underbrace{l_{e,2}}_{l_2} = L(\, x_2 \,|\, y_1, y_3 \,) = \ln \left(\frac{\Pr(\, x_2 = 0 \,|\, y_1, y_3 \,)}{\Pr(\, x_2 = 1 \,|\, y_1, y_3 \,)} \right) \tag{4.58}$$

By using equation (4.51) and (4.56) (4.59)

$$= \ln \frac{\alpha \cdot \Pr(\, y_1 \,|\, X_1 = 0 \,) \cdot \Pr(\, y_3 \,|\, X_3 = 0 \,)}{\alpha \cdot \Pr(\, y_1 \,|\, X_1 = 1 \,) \cdot \Pr(\, y_3 \,|\, X_3 = 1 \,)} \tag{4.60}$$

$$= \underbrace{\ln \frac{\Pr(\, y_1 \,|\, X_1 = 0 \,)}{\Pr(\, y_1 \,|\, X_1 = 1 \,)}}_{l_{ch,1}} + \underbrace{\ln \frac{\Pr(\, y_3 \,|\, X_3 = 0 \,)}{\Pr(\, y_3 \,|\, X_3 = 1 \,)}}_{l_{ch,3}} \tag{4.61}$$

$$l_{e,2} = \underbrace{l_{ch,1}}_{l_1} + \underbrace{l_{ch,3}}_{l_3} \tag{4.62}$$

Summary of the message update rules for a variable node degree three

It should be noted that there will always be at least one intrinsic message from the channel and the rest incoming extrinsic messages are from the neighbouring check nodes. Moreover, if the variable node has degree (d_v) 2, then the only incoming message from the channel is equal to the outgoing extrinsic message. However, if the variable node has degree at least (d_v) 3 (see Figure 4.5), the general extrinsic message update rules can be shown in terms of probabilities and L-values.

The update rules are generalized and summarized for a variable node having degree 3 using equations (4.52) and (4.57) for messages in terms of probabilities and (4.61) for messages in terms of L-values.

Here new notations (VAR and CHK) are also introduced [9] which can also be used for the generalization of the update rules easily. In the Figure 4.5 the two incoming messages are $(\mu_1,\ \mu_3)$ or $(l_1,\ l_3)$ and the outgoing message is μ_2 or l_2. These notations can be used in such a way,

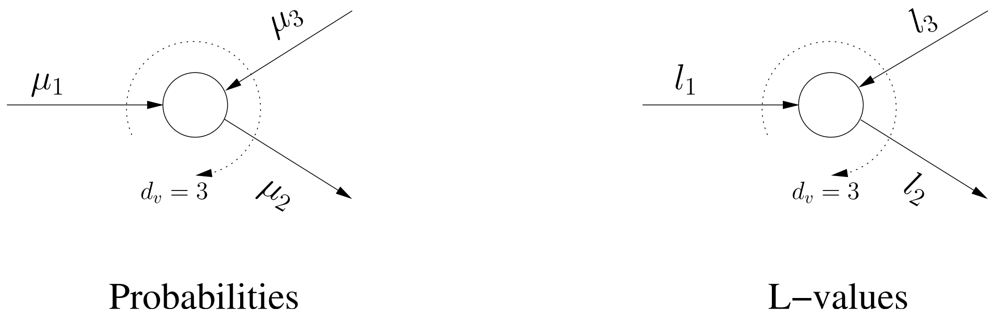


Figure 4.5: *Variable node having degree (d_v) 3, the outgoing extrinsic message is μ_2 (l_2) and the two incoming messages are μ_1 (l_1) and μ_3 (l_3).*

In terms of probabilities:

$$\begin{pmatrix} \mu_2(0) \\ \mu_2(1) \end{pmatrix} = VAR\big(\mu_1,\ \mu_3\big) = \begin{pmatrix} \alpha \cdot \mu_1(0)\mu_3(0) \\ \alpha \cdot \mu_1(1)\mu_3(1) \end{pmatrix} \tag{4.63}$$

where, α is a scaling factor such that $\mu_2(0) + \mu_2(1) = 1$.

In terms of L-values:

$$l_2 = VAR\big(l_1,\ l_3\big) = l_1 + l_3 \tag{4.64}$$

It can be shown that these L-values at the variable node side can be scaled by any constant β.

$$\beta \cdot l_2 = \beta \cdot VAR\big(l_1,\ l_3\big) = \beta \cdot l_1 + \beta \cdot l_3 \tag{4.65}$$

Generalization of the update rules for a variable node of any degree

The repetition code of length N can be pondered over after taking the repetition code of length 3. The generalization can easily be seen because the repetition code has always two codewords either all-zeros or all-ones. Therefore, in the sum-product formulation of both extrinsic and a-posteriori there will be no summation over both codewords when one code symbol is fixed and in terms of log likelihood ratios (LLR or L-values), the formulation can be shown as, *Extrinsic L-value of the repetition code having length N* can be found in such a way, (4.32) and (4.33)

$$l_{e,i} = \ln\left(\frac{p_{e,i}^{0}}{p_{e,i}^{1}}\right) \tag{4.66}$$

$$= \ln\left(\frac{\alpha \cdot \sum\limits_{\substack{\boldsymbol{x}\in\mathcal{C}\\ x_i=0}} \prod\limits_{\substack{j=1\\ j\neq i}}^{N} p_{ch,j}^{x_j}}{\alpha \cdot \sum\limits_{\substack{\boldsymbol{x}\in\mathcal{C}\\ x_i=1}} \prod\limits_{\substack{j=1\\ j\neq i}}^{N} p_{ch,j}^{x_j}}\right) \tag{4.67}$$

$$= \ln\left(\frac{\sum\limits_{\boldsymbol{x}=\underbrace{\{0,0,\ldots 0\}}_{\text{length } N}} \prod\limits_{\substack{j=1\\ j\neq i}}^{N} p_{ch,j}^{x_j}}{\sum\limits_{\boldsymbol{x}=\underbrace{\{1,1,\ldots 1\}}_{\text{length } N}} \prod\limits_{\substack{j=1\\ j\neq i}}^{N} p_{ch,j}^{x_j}}\right) \tag{4.68}$$

$$= \ln\left(\prod_{\substack{j=1\\ j\neq i}}^{N} \left(\frac{p_{ch,j}^{0}}{p_{ch,j}^{1}}\right)\right) \tag{4.69}$$

$$= \sum_{\substack{j=1\\ j\neq i}}^{N} \ln\left(\frac{p_{ch,j}^{0}}{p_{ch,j}^{1}}\right) \tag{4.70}$$

hence,

$$l_{e,i} = \sum_{\substack{j=1\\ j\neq i}}^{N} l_{ch,j} \tag{4.71}$$

Similarly, *A-posteriori L-value of the repetition code having length* N can be found in the same way as for extrinsic L-value using equations (4.24) and (4.25).

$$l_{p,i} = \sum_{j=1}^{N} \ln\left(\frac{p^0_{ch,j}}{p^1_{ch,j}}\right) \tag{4.72}$$

$$= \sum_{j=1}^{N} l_{ch,j} \tag{4.73}$$

$$= l_{ch,i} + l_{e,i} \tag{4.74}$$

Finally, it can be construed from the property of the extrinsic L-value of the repetition code having length N that the all incoming L-values, i.e., at least one incoming L-value from the channel and the rest from the neighbouring check nodes, will be summed up to render extrinsic or a-posteriori L-value.

The proof of the message update rules of a variable node having degree (d_v) is not shown, but it is just summarized and generalized from [9]. Figure 4.6 shows a variable node of any cycle free factor graph. However, the generalized update rules can be proved by induction. In this generalized figure, a variable node n of any factor graph represented by check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M\times N}$. The extrinsic outgoing message, say $\mu_{vc}(n, D)$, needs $d_v - 1$ incoming messages at the variable node n, so the update rule can be generalized in the following way,

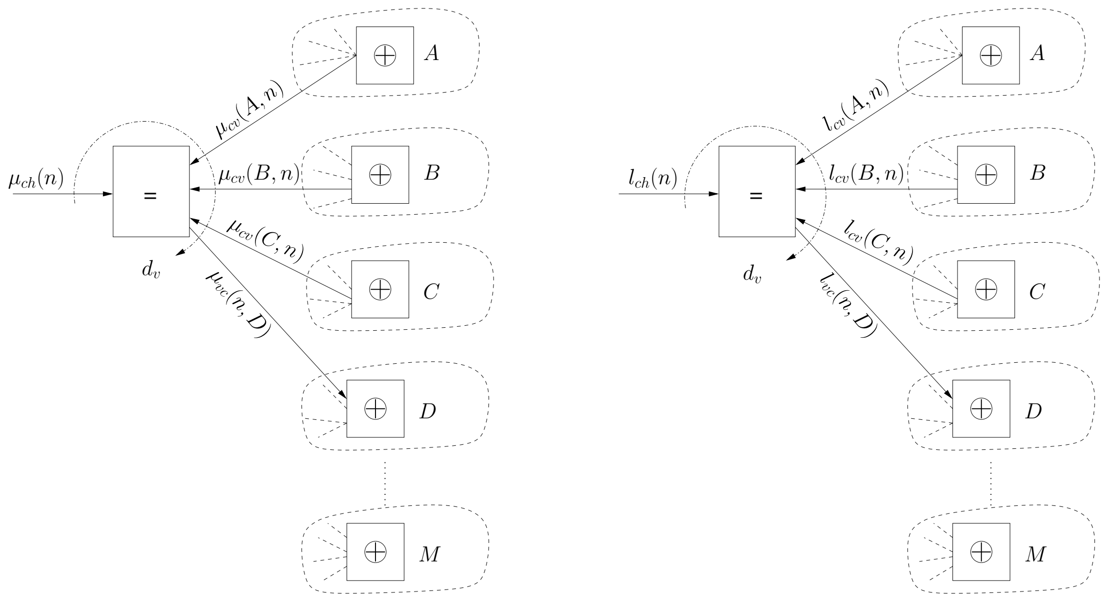


Figure 4.6: *The general view of a variable node of any cycle free factor graph.*

Extrinsic outgoing message:

In terms of probabilities:

$$\mu_{vc}(n, D) = VAR(\underbrace{\mu_{ch}(n),\ \mu_{cv}(A, n),\ \mu_{cv}(B, n),\ \mu_{cv}(C, n),\ \dots}_{(d_v - 1)\text{ messages}}) \tag{4.75}$$

$$= VAR\Bigg(\ \mu_{ch}(n),\ VAR\Big(\mu_{cv}(A, n),\ \mu_{cv}(B, n),\ \mu_{cv}(C, n),\ \dots\Big)\ \Bigg) \tag{4.76}$$

Similarly, in terms of L-values:

$$l_{vc}(n, D) = VAR(\underbrace{l_{ch}(n),\ l_{cv}(A, n),\ l_{cv}(B, n),\ l_{cv}(C, n),\ \dots}_{(d_v - 1)\text{ L-values}}) \tag{4.77}$$

$$= VAR\Bigg(\ l_{ch}(n),\ VAR\Big(l_{cv}(A, n),\ l_{cv}(B, n),\ l_{cv}(C, n),\ \dots\Big)\ \Bigg) \tag{4.78}$$

$$= l_{ch}(n) + l_{cv}(A, n) + l_{cv}(B, n) + l_{cv}(C, n) + \ \dots \tag{4.79}$$

4.3.5.2 Message Update Rules for Check Nodes

The message update rules for check nodes can be derived in a similar way as for variable nodes. The same example can be considered whose Forney factor graph is shown in the Figure 4.2. The independent and isolated check node A of this figure is shown in the Figure 4.7.

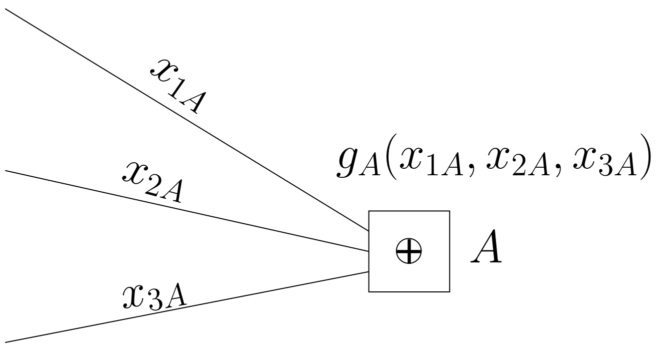


Figure 4.7: *The check node A of the Figure 4.2 is considered to show the single parity check code constraint.*

The indicator function of a check node A is defined as $g_A(x_{1A},\ x_{2A},\ x_{3A})$ [11] where x_{1A}, x_{2A} and x_{3A} are the variables, such that

$$g_A(x_{1A},\ x_{2A},\ x_{3A}) = \begin{cases} 1, & \text{if } x_{1A} \oplus x_{2A} \oplus x_{3A} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.80}$$

So, the equation (4.80) implies that all the edges which are connected to a check node must fulfill the parity constraint i.e., $x_{1A} \oplus x_{2A} \oplus x_{3A} = 0$. Thus, the message update rules of the check nodes can be defined considering the check node has got the single parity check code constraint.

Now an example of a single parity check code length $N = 3$ can be considered to explain the message update rules for the check nodes having degree (d_c) 3. Thereafter, the message update rules for check nodes having degree more than 3 can also be generalized.

A single parity check code having length 3 is assumed such that $\mathcal{C} = \{000, 011, 101, 110\}$. The codeword $\boldsymbol{x} = [x_1\ x_2\ x_3]^T \in \mathcal{C}$ is transmitted across the memoryless channel and the received word is $\boldsymbol{y} = [y_1\ y_2\ y_3]^T$. The single parity check code is represented by the following check matrix whose Forney factor graph is shown in the Figure 4.8.

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{4.81}$$

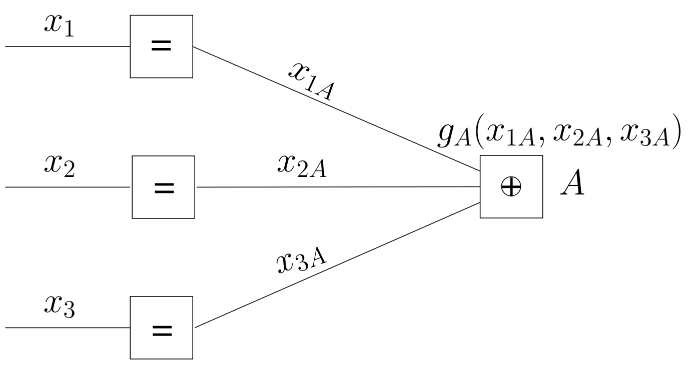


Figure 4.8: *The Forney factor graph of a single parity check code having length* 3.

The extrinsic message of the code symbol x_2 is considered to explain the update rules for the single parity check code of length $N = 3$. So, from equation (4.32) and (4.33),

$$\Pr(\, x_2 = 0 \,|\, y_{\backslash 2} \,) = \Pr(\, x_2 = 0 \,|\, y_1,\; y_3 \,) = \underbrace{p^0_{e,2}}_{\mu_2(0)} \tag{4.82}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_2 = \underline{0}}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j \,) \tag{4.83}$$

$$= \alpha \cdot \sum_{\boldsymbol{x} = \{0\underline{0}0,\; 1\underline{0}1\}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j \,) \tag{4.84}$$

$$\begin{aligned} &= \alpha \cdot \left\{ \sum_{\boldsymbol{x} = \{0\underline{0}0\}} \Pr(\, y_1 \,|\, X_1 = 0 \,) \cdot \Pr(\, y_3 \,|\, X_3 = 0 \,) \right\} + \\ &+ \alpha \cdot \left\{ \sum_{\boldsymbol{x} = \{1\underline{0}1\}} \Pr(\, y_1 \,|\, X_1 = 1 \,) \cdot \Pr(\, y_3 \,|\, X_3 = 1 \,) \right\} \end{aligned} \tag{4.85}$$

$$p^0_{e,2} = \alpha \cdot \underbrace{p^0_{ch,1}}_{\mu_1(0)} \cdot \underbrace{p^0_{ch,3}}_{\mu_3(0)} + \alpha \cdot \underbrace{p^1_{ch,1}}_{\mu_1(1)} \cdot \underbrace{p^1_{ch,3}}_{\mu_3(1)} \tag{4.86}$$

Similarly,

$$\Pr(\, x_2 = 1 \,|\, y_{\backslash 2} \,) = \Pr(\, x_2 = 1 \,|\, y_1,\; y_3 \,) = \underbrace{p^1_{e,2}}_{\mu_2(1)} \tag{4.87}$$

$$= \alpha \cdot \sum_{\substack{\boldsymbol{x} \in \mathcal{C} \\ x_2 = \underline{1}}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j \,) \tag{4.88}$$

$$p_{e,2}^{1} = \alpha \cdot \sum_{\boldsymbol{x} = \{0\underline{1}1,\ 1\underline{1}0\}} \prod_{\substack{j=1 \\ j \neq 2}}^{N=3} \Pr(\, y_j \,|\, X_j = x_j \,) \tag{4.89}$$

$$\begin{aligned} &= \alpha \cdot \left\{ \sum_{\boldsymbol{x} = \{0\underline{1}1\}} \Pr(\, y_1 \,|\, X_1 = 0\,) \cdot \Pr(\, y_3 \,|\, X_3 = 1\,) \right\} + \\ &+ \alpha \cdot \left\{ \sum_{\boldsymbol{x} = \{1\underline{1}0\}} \Pr(\, y_1 \,|\, X_1 = 1\,) \cdot \Pr(\, y_3 \,|\, X_3 = 0\,) \right\} \end{aligned} \tag{4.90}$$

$$p_{e,2}^{1} = \alpha \cdot \underbrace{p_{ch,1}^{0}}_{\mu_1(0)} \cdot \underbrace{p_{ch,3}^{1}}_{\mu_3(1)} + \alpha \cdot \underbrace{p_{ch,1}^{1}}_{\mu_1(1)} \cdot \underbrace{p_{ch,3}^{0}}_{\mu_3(0)} \tag{4.91}$$

where, the scaling factor α can be found such that $p_{e,2}^{0} + p_{e,2}^{1} = 1$.

In terms of log likelihood ratios (L-values)

From equations (4.86) and (4.91), the extrinsic message can be derived as,

$$\underbrace{L(\, x_2 \,|\, y_1, y_3 \,)}_{l_2} = \ln\left(\frac{\Pr(\, x_2 = 0 \,|\, y_1, y_3 \,)}{\Pr(\, x_2 = 1 \,|\, y_1, y_3 \,)}\right) = \ln\left(\frac{p_{e,2}^{0}}{p_{e,2}^{1}}\right) \tag{4.92}$$

$$= \ln\left(\frac{p_{ch,1}^{0} \cdot p_{ch,3}^{0} \; + \; p_{ch,1}^{1} \cdot p_{ch,3}^{1}}{p_{ch,1}^{0} \cdot p_{ch,3}^{1} \; + \; p_{ch,1}^{1} \cdot p_{ch,3}^{0}}\right) \tag{4.93}$$

$$= l_1 \boxplus l_3 \qquad \text{See Appendix B for the proof} \tag{4.94}$$

$$= 2\tanh^{-1}\left(\tanh\left(\frac{l_1}{2}\right) \cdot \tanh\left(\frac{l_3}{2}\right)\right) \tag{4.95}$$

It can be seen that the L-values at the check node side of the sum-product algorithm can not be scaled by any constant β.

$$\beta \cdot l_2 \neq \beta \cdot l_1 \boxplus \beta \cdot l_3 \tag{4.96}$$

Summary of the message update rules for a check node degree three

In the above example of single parity check code having length 3, the local decoding operation performed by the check node renders box-plus summation on incoming L-values from the neighbouring variable nodes. The meaning of the box-plus of L-values can be seen as the transform of modulo 2 addition into box-plus of L-values which is caused by the property of the single parity check code constraint at the check node, i.e.,

$$x_1 \oplus x_2 \oplus x_3 = 0 \tag{4.97}$$

$$\Leftrightarrow x_2 = x_1 \oplus x_3 \tag{4.98}$$

$$\Rightarrow L(x_2) = L(x_1) \boxplus L(x_3) \tag{4.99}$$

where, $x_1, x_2, x_3 \in \mathbb{F}_2$. The update rules are summarized for a check node having degree (d_c) 3 using equations (4.86) and (4.91) for messages in terms of probabilities and (4.94) for messages in terms of L-values. In the Figure 4.9 the two incoming messages are $(\mu_1,\ \mu_3)$ or $(l_1,\ l_3)$ and the outgoing message is μ_2 or l_2. The notation/function CHK can be used in such a way,

In terms of probabilities:

$$\begin{pmatrix} \mu_2(0) \\ \mu_2(1) \end{pmatrix} = CHK\big(\mu_1,\ \mu_3\big) = \alpha \cdot \begin{pmatrix} \mu_1(0)\mu_3(0) + \mu_1(1)\mu_3(1) \\ \mu_1(0)\mu_3(1) + \mu_1(1)\mu_3(0) \end{pmatrix} \tag{4.100}$$

where, α is a scaling factor such that $\mu_2(0) + \mu_2(1) = 1$.

In terms of L-values:

$$l_2 = CHK\big(l_1,\ l_3\big) = l_1 \boxplus l_3 = 2\tanh^{-1}\left(\tanh\left(\frac{l_1}{2}\right) \cdot \tanh\left(\frac{l_3}{2}\right)\right) \tag{4.101}$$

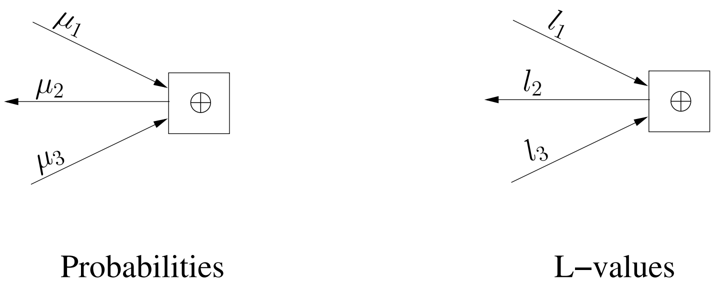


Figure 4.9: *Check node having degree (d_c) 3, the outgoing extrinsic message is μ_2 (l_2) and the two incoming messages are μ_1 (l_1) and μ_3 (l_3).*

Generalization of the update rules for a check node

Similar to variable node, the proof of the message update rules of a check node having degree (d_c) is not shown but it is summarized and generalized from [9]. Figure 4.10 shows a check node of any cycle free factor graph. In this generalized figure, the check node m of any factor graph represented by check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M\times N}$. The extrinsic outgoing message, say $\mu_{cv}(m,2)$, needs $d_c - 1$ incoming messages at the check node n, so the update rule can be generalized in the following way,

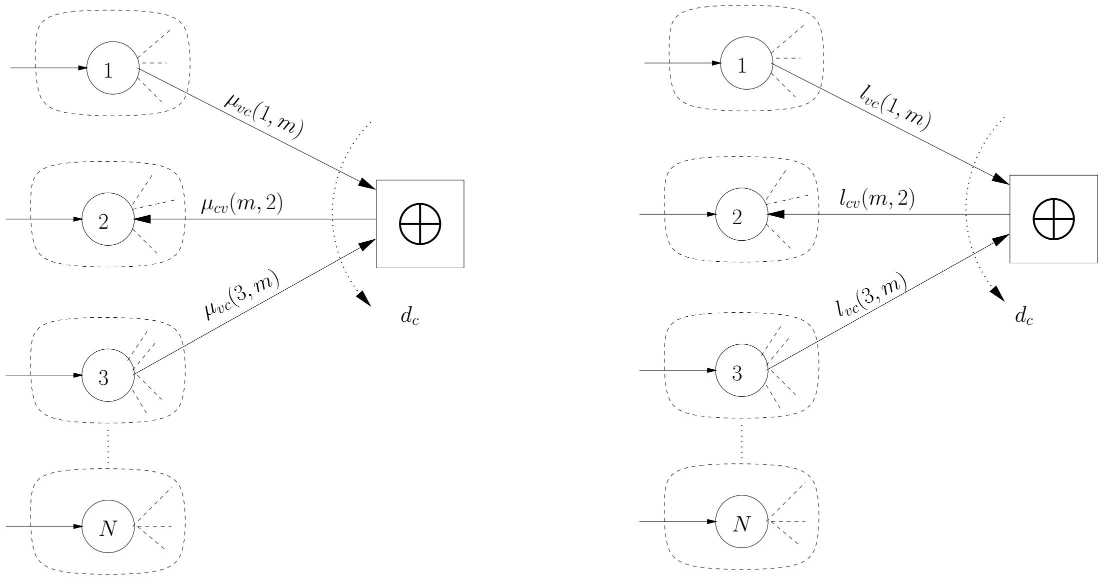


Figure 4.10: *The general view of a check node of any factor graph.*

Extrinsic outgoing message:

In terms of probabilities:

$$\mu_{cv}(m,2) = CHK\Big(\underbrace{\mu_{vc}(1,m),\ \mu_{vc}(3,m),\ \ldots}_{(d_c-1)\text{ messages}}\Big) \tag{4.102}$$

$$= CHK\Bigg(\mu_{vc}(1,m),\ CHK\Big(\mu_{vc}(3,m)\ \ldots\Big)\Bigg) \tag{4.103}$$

Similarly, in terms of L-values:

$$l_{cv}(m,2) = CHK\Big(\underbrace{l_{vc}(1,m),\ l_{vc}(3,m),\ \dots}_{(d_c-1)\text{ L-values}}\Big) \tag{4.104}$$

$$= CHK\Bigg(l_{vc}(1,m),\ CHK\Big(l_{vc}(3,m)\ \dots\Big)\Bigg) \tag{4.105}$$

$$= l_{vc}(1,m)\ \boxplus\ l_{vc}(3,m)\ \boxplus\ \dots \tag{4.106}$$

4.3.6 Example for Sum-Product Algorithm

The message update rules for the sum-product algorithm has been shown in the previous sections. Now, a small example is considered to show the process of decoding by the sum-product algorithm such that the update rules of a variable node and check node can be used to show the concepts of decoding.

A simple repetition code $\mathcal{C} = \big\{00000, 11111\big\}$ is considered which is represented in the Figure 4.11 with the channel coding model. It is assumed that a codeword $\boldsymbol{x} = [x_1\, x_2\, x_3\, x_4\, x_5]^T \in \mathcal{C}$ is transmitted across the BSC with the cross over probability $\epsilon = 0.2$ (cf. section 2.2 for BSC). So, the probabilities can be defined as

$$\Pr(\, y_n \,|\, x_n\,) = \begin{cases} 0.8, & \text{if } y_n = x_n \\ 0.2, & \text{if } y_n \neq x_n \end{cases} \tag{4.107}$$

for $n = 1,2,3,4,5$. It is assumed that the received word is $\boldsymbol{y} = [y_1\ y_2\ y_3\ y_4\ y_5]^T = [0\ 0\ 1\ 0\ 0]^T$. The messages in terms of probabilities are shown in the Figure 4.11 and 4.12 as $\begin{pmatrix} \mu_n(0) \\ \mu_n(1) \end{pmatrix}$ such that they can be scaled in order to be $\mu_n(0) + \mu_n(1) = 1$.

The received symbol 0 is represented as the message $\begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}$ while 1 is represented as the message $\begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}$.

Initially, the received messages at the variable nodes are sent along the edges to the neighbouring check nodes which is shown in the Figure 4.12 (i). Then, check nodes compute the outgoing extrinsic messages (for update rules, see Page 41 and 49), for example the outgoing message $\mu_{cv}(A,1)$ from check node A to variable node 1 is

$$\mu_{cv}(A,1) = CHK\left(\begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}, \begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}\right) = \begin{pmatrix} 0.8\cdot 0.2\ +\ 0.8\cdot 0.2 = 0.32 \\ 0.8\cdot 0.8\ +\ 0.2\cdot 0.2 = 0.68 \end{pmatrix} \tag{4.108}$$

Similarly, the rest 5 outgoing messages are calculated which are shown in the Figure 4.12 (ii).

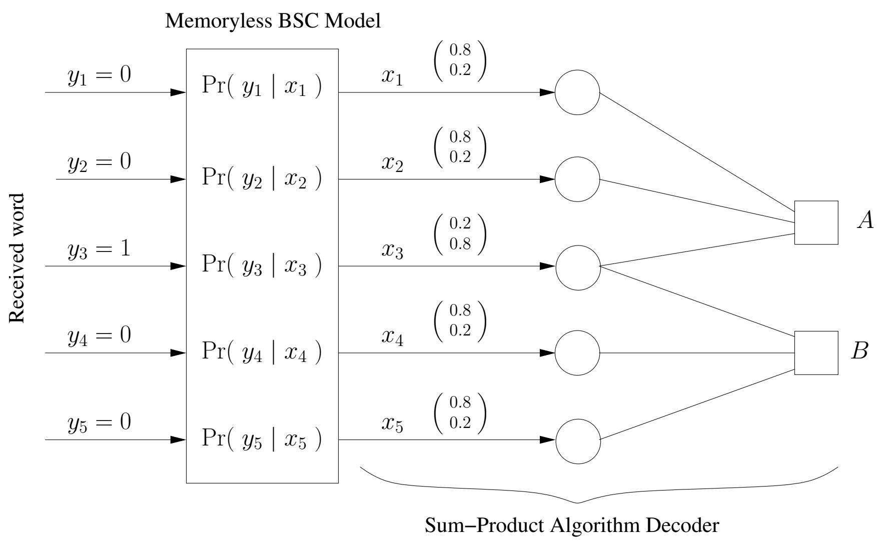

Memoryless BSC Model
Received word
$y_1 = 0$
$y_2 = 0$
$y_3 = 1$
$y_4 = 0$
$y_5 = 0$
Pr($y_1 \mid x_1$)
Pr($y_2 \mid x_2$)
Pr($y_3 \mid x_3$)
Pr($y_4 \mid x_4$)
Pr($y_5 \mid x_5$)
x_1 $\begin{pmatrix}0.8\\0.2\end{pmatrix}$
x_2 $\begin{pmatrix}0.8\\0.2\end{pmatrix}$
x_3 $\begin{pmatrix}0.2\\0.8\end{pmatrix}$
x_4 $\begin{pmatrix}0.8\\0.2\end{pmatrix}$
x_5 $\begin{pmatrix}0.8\\0.2\end{pmatrix}$
A
B
Sum−Product Algorithm Decoder

Figure 4.11: *The factor graph of a repetition code having length* 5 *with the coding model is shown. The received word is* $\boldsymbol{y} = [0\;0\;1\;0\;0]^T$.

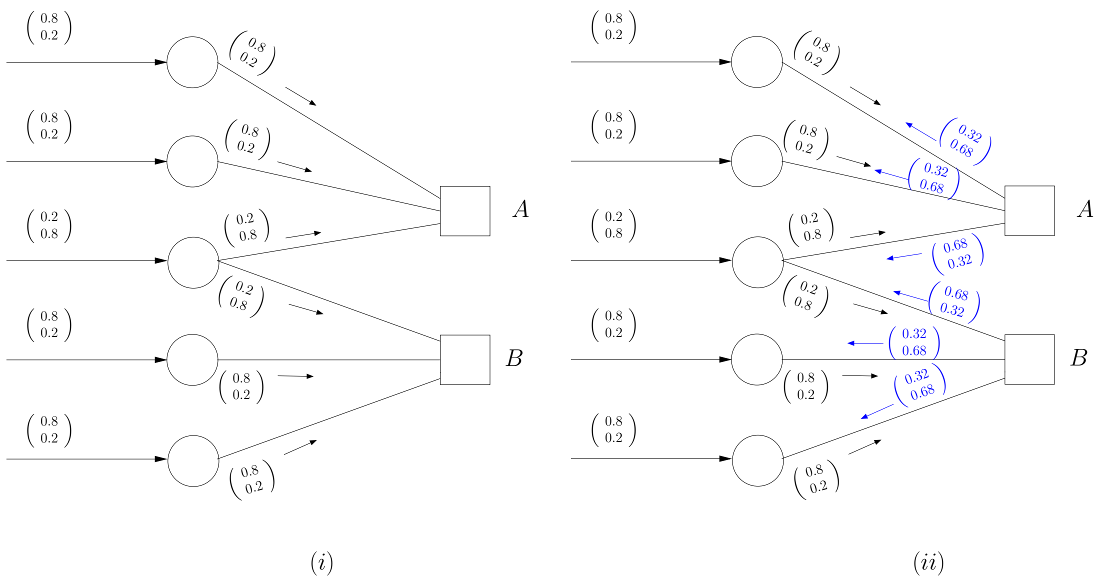


Figure 4.12: (i) *Initially, the messages received from the channel at the variable nodes are passed to the neighbouring check nodes;* (ii) *Representing the outgoing extrinsic messages out of the check nodes in blue colour.*

Now, before the termination of the one complete iteration, the variable nodes have to compute the new messages $\mu_v(n)$ to check whether the estimated codeword $\hat{\boldsymbol{x}}$ which is formed by stacking all 5 code symbols in a vector fulfills $\underline{\mathbf{H}} \otimes \hat{\boldsymbol{x}} = 0$. For example, the variable node 3 computes the message $\mu_v(3)$ as

$$\mu_v(3) = VAR\left(\begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}, \begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix}, \begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix}\right) \tag{4.109}$$

$$= VAR\left(\begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}, VAR\left\{\begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix}, \begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix}\right\}\right) \tag{4.110}$$

$$= VAR\left(\begin{pmatrix} 0.2 \\ 0.8 \end{pmatrix}, \begin{pmatrix} \alpha \cdot 0.68 \cdot 0.68 = \alpha \cdot 0.4624 \\ \alpha \cdot 0.32 \cdot 0.32 = \alpha \cdot 0.1024 \end{pmatrix}\right) \tag{4.111}$$

$$= \begin{pmatrix} 0.2 \cdot \alpha \cdot 0.4624 = \alpha \cdot 0.09248 \\ 0.8 \cdot \alpha \cdot 0.1024 = \alpha \cdot 0.08192 \end{pmatrix} = \begin{pmatrix} \mu_3(0) \\ \mu_3(1) \end{pmatrix} \tag{4.112}$$

where α is a scaling factor.

By the hard-decision of each message $[\mu_v(1), \mu_v(2), \mu_v(3), \mu_v(4), \mu_v(5)]$, each code symbols can be estimated. So, the hard-decision of $\mu_v(n)$ can be done in this way,

$$\hat{x}_n = \begin{cases} 0, & \text{if } \mu_n(0) \geq \mu_n(1) \\ 1, & \text{if } \mu_n(0) < \mu_n(1) \end{cases} \tag{4.113}$$

For example, for $\mu_v(3): \ \mu_3(0) = \alpha \cdot 0.09248, \ \mu_3(1) = \alpha \cdot 0.08192$, so the code symbol $\hat{x}_3 = 0$ because $\mu_3(0) \geq \mu_3(1)$. Thus, the estimated codeword $\hat{\boldsymbol{x}}$ comes out to be $[0\ 0\ 0\ 0\ 0]^T$ by hard-decisions. If it fulfills the $\underline{\mathbf{H}} \otimes \hat{\boldsymbol{x}} = \mathbf{0}$, then the iteration will be terminated. Hence, the received word is decoded successfully and the iteration is terminated.

4.4 Max-Product / Min-Sum Algorithm

It can be accentuated here that if the "sum" are replaced by the "max" everywhere in the sum-product formulation then the variant will be max-product algorithm. Moreover, the sum-product algorithm decodes symbol-wise while the max-product decodes block-wise.

Similar to the sum-product formulation derivation, if a code $\mathcal{C}$ which is defined by a parity check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M \times N}, \ M \geq N - K$ is considered such that $\mathcal{C} \in \mathbb{F}_2^N$, where $|\mathcal{C}| = 2^K$, information word length K and code rate $R = \frac{K}{N}$. If a codeword from the code is transmitted across the memoryless channel, then the max-product algorithm can be used to decode the received word. If the factor graph of this code is cycle-free then the decoding will be maximum likelihood (optimal) otherwise for the non-cycle free, the decoding will be sub-optimal. The assumptions are same that the code symbols and codewords in a code are equiprobable.

Formally,

$$\begin{aligned}\mathcal{C} &= \Big\{\boldsymbol{x} : \boldsymbol{x} = enc(\boldsymbol{u}),\ \text{information word } \boldsymbol{u} \in \mathbb{F}_2^K\Big\}\\ &= \Big\{\boldsymbol{x} \in \mathbb{F}_2^N\ :\ \underline{\mathbf{H}} \otimes \boldsymbol{x} = \mathbf{0}\Big\}\end{aligned}$$

The max-product formulation can be derived, if "sum" are replaced by the "max" everywhere in the sum-product formulation (4.20) such that

$$x_n = \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \underbrace{\underbrace{\max_{\substack{\boldsymbol{x} \in \mathcal{C}\\ x_n \text{ fixed}}}}_{Max} \underbrace{\prod_{j=1}^{N}}_{Product} \Pr(\,y_j\,|\,x_j\,)}_{\max \Pr(\,x_n\,|\,y\,) = \max APP} \tag{4.114}$$

$$= \underset{x_n \in \{0,1\}}{\operatorname{argmax}} \max_{\substack{\boldsymbol{x} \in \mathcal{C}\\ x_n \text{ fixed}}} \Pr(\,\boldsymbol{y}\,|\,\boldsymbol{x}\,) \tag{4.115}$$

It is very conspicuous from the equation (4.115) that the max-product algorithm performs decoding ML-block wise, i.e.,

$$\hat{\boldsymbol{x}}_{ML} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmax}}\ \Pr(\,\boldsymbol{y}\,|\,\boldsymbol{x}\,) \tag{4.116}$$

Hence, it can be reiterated that the max-product decodes block-wise while sum-product decodes symbol-wise optimally for cycle free factor graphs. In terms of L-values, the max-product algorithm is called as the min-sum algorithm. It is shown in this section that the the approximation of the box-plus operator in the sum-product algorithm renders min-sum algorithm [9] [10].

It can be said in other words that the message update rules for the max-product algorithm at the variable node side is similar to the sum-product algorithm because in the sum-product formulation there is no "sum", and hence both of them have the same update rules at the variable node side. However, the max-product has different update rules at the check node side because the "sum" is replaced by the "max" everywhere. So, the max-product/min-sum algorithm has different update rules for the check node.

It can be possible that the sum-product algorithm may estimate a codeword which does not exist in the code (cycle or cycle-free) specified at the encoder and decoder because sum-product decodes symbol-wise. But, the max-product algorithm will always estimate a codeword from the code (cycle or cycle-free) because max-product decodes block-wise. However, the max-product/min-sum algorithm's performance is bit worse (higher word-error rate) than the sum-product algorithm for the same signal to noise ratio (SNR) [1].

4.4.1 Update Rules of Max-Product/Min-Sum Algorithm

The message update rules of the max-product/min-sum algorithm can be summarized and derived for any variable and check node having degree d_v and d_c respectively of a cycle free factor graph.

1. The message update rules of the max-product algorithm for a variable node are exactly same as to the sum-product algorithm. The update rules of the max-product algorithm are summarized and reiterated for a variable node degree 3 and any degree d_v.

   ◦ For a variable node having degree $d_v = 3$, see Figure 4.5.

   In terms of probabilities:

$$\begin{pmatrix} \mu_2(0) \\ \mu_2(1) \end{pmatrix} = VAR\big(\mu_1,\ \mu_3\big) = \begin{pmatrix} \alpha \cdot \mu_1(0)\mu_3(0) \\ \alpha \cdot \mu_1(1)\mu_3(1) \end{pmatrix} \tag{4.117}$$

(4.118)

   where, α is a scaling factor.

   In terms of L-values:

$$l_2 = VAR\big(l_1,\ l_3\big) = l_1 + l_3 \tag{4.119}$$

$$\beta \cdot l_2 = \beta \cdot VAR\big(l_1,\ l_3\big) = \beta \cdot l_1\ +\ \beta \cdot l_3 \tag{4.120}$$

   where, μ_2 (l_2) is outgoing extrinsic message and μ_1 (l_1) and μ_3 (l_3) are incoming extrinsic messages. The equation 4.120 shows that the L-values can be scaled by a constant β which implies that the knowledge of signal to noise ratio is not important (cf. section 2.5).

- For a variable node having any degree d_v, see Figure 4.6.

  Extrinsic outgoing message:

  In terms of probabilities:

$$\mu_{vc}(n, D) = VAR\big(\underbrace{\mu_{ch}(n),\ \mu_{cv}(A, n),\ \mu_{cv}(B, n),\ \mu_{cv}(C, n),\ \dots}_{(d_v - 1)\text{ messages}}\big) \tag{4.121}$$

$$= VAR\Bigg(\ \mu_{ch}(n),\ VAR\Big(\mu_{cv}(A, n),\ \mu_{cv}(B, n),\ \mu_{cv}(C, n),\ \dots\Big)\Bigg) \tag{4.122}$$

  Similarly, in terms of L-values:

$$l_{vc}(n, D) = VAR\big(\underbrace{l_{ch}(n),\ l_{cv}(A, n),\ l_{cv}(B, n),\ l_{cv}(C, n),\ \dots}_{(d_v - 1)\text{ L-values}}\big) \tag{4.123}$$

$$= VAR\Bigg(\ l_{ch}(n),\ VAR\Big(l_{cv}(A, n),\ l_{cv}(B, n),\ l_{cv}(C, n),\ \dots\Big)\Bigg) \tag{4.124}$$

$$= l_{ch}(n) + l_{cv}(A, n) + l_{cv}(B, n) + l_{cv}(C, n) + \ \dots \tag{4.125}$$

2. The message update rules of the max-product algorithm for a check node are different from the sum-product algorithm. As it was discoursed before, that the max-product / min-sum algorithm finds the maximum of the APP in terms of probabilities and approximates the box-plus in terms of L-values (LLR).

   - For a check node degree having $d_c = 3$, see Figure 4.9, where two incoming messages are $(\mu_1,\ \mu_3)$ or $(l_1,\ l_3)$ and the outgoing message is μ_2 or l_2. Then,

     In terms of probabilities:

$$\begin{pmatrix} \mu_2(0) \\ \mu_2(1) \end{pmatrix} = CHK\big(\mu_1,\ \mu_3\big) = \alpha \cdot \begin{pmatrix} \max\{\mu_1(0)\mu_3(0),\ \mu_1(1)\mu_3(1)\} \\ \max\{\mu_1(0)\mu_3(1),\ \mu_1(1)\mu_3(0)\} \end{pmatrix} \tag{4.126}$$

     where, α is a scaling factor. It should be noticed that in the update rules of the sum-product algorithm the "sum" is replaced by "max" rendering max-product algorithm.

In terms of L-values:

In the sum-product algorithm, the update rules for a check node degree 3 were

$$l_2 = CHK\big(l_1,\ l_3\big) = l_1 \boxplus l_3 = 2\tanh^{\text{-1}}\left(\tanh\left(\frac{l_1}{2}\right)\cdot\tanh\left(\frac{l_3}{2}\right)\right) \tag{4.127}$$

$$= \ln\left(\frac{\cosh\left(\left(l_1+l_3\right)/2\right)}{\cosh\left(\left(l_1-l_3\right)/2\right)}\right) \qquad \text{(see Appendix B for proof)} \tag{4.128}$$

$$= \ln\left(\cosh\left(\left(l_1+l_3\right)/2\right)\right)\ -\ \ln\left(\cosh\left(\left(l_1-l_3\right)/2\right)\right) \tag{4.129}$$

It is known [9] that

$$\ln\Big(\cosh(Z)\Big) \approx |Z| - \ln(2) \qquad \text{for } Z \gg 1 \tag{4.130}$$

Then, continuing equation (4.129),

$$CHK\big(l_1,\ l_3\big) \approx \left|\left(l_1+l_3\right)/2\right|\ -\ \left|\left(l_1-l_3\right)/2\right| \tag{4.131}$$

$$= sgn(l_1)\ sgn(l_3)\ \min\left(\left|l_1\right|,\ \left|l_3\right|\right) \tag{4.132}$$

So, it turns out that the approximation of the box-plus operator in the sum-product algorithm is called as min-sum algorithm [9].

It can be seen that the L-values of the min-sum algorithm can be scaled by any constant β.

$$\beta\cdot l_2 = \beta\cdot CHK\big(l_1,\ l_3\big) = sgn(\beta\cdot l_1)\ sgn(\beta\cdot l_3)\ \min\left(\left|\beta\cdot l_1\right|,\ \left|\beta\cdot l_3\right|\right) \tag{4.133}$$

- For a check node having any degree d_c, see Figure 4.10

Extrinsic outgoing message:

In terms of probabilities:

$$\mu_{cv}(m,2) = CHK\Big(\underbrace{\mu_{vc}(1,m),\ \mu_{vc}(3,m),\ \dots}_{(d_c-1)\text{ messages}}\Big) \tag{4.134}$$

$$= CHK\Big(\ \mu_{vc}(1,m),\ CHK\big(\mu_{vc}(3,m)\ \dots\big)\Big) \tag{4.135}$$

Similarly, in terms of L-values:

$$l_{cv}(m,2) = CHK\Big(\underbrace{l_{vc}(1,m),\ l_{vc}(3,m),\ \ldots}_{(d_c-1)\ \text{L-values}}\Big) \tag{4.136}$$

$$= CHK\Big(l_{vc}(1,m),\ CHK\big(l_{vc}(3,m)\ \ldots\ l_{d_c-2},\ l_{d_c-1}\big)\Big) \tag{4.137}$$

Before getting further, it shall be noted that the the sum-product and min-sum algorithms were implemented using L-values to decode a $(7,4,3)$ Hamming code under BSC in MATLAB (cf. the MATLAB code named as *test_mpa.m* in the CD-ROM attached with the report).

4.5 Message Passing Algorithm for the BEC

The sum-product and max-product/min-sum algorithm has been described in the previous sections. If the channel is presumed to be Binary Erasure Channel (BEC), then the performance of the MPA can easily be described without considering probabilistic decoding for update rules.

4.5.1 Node Operations and Algorithm

Initially, a code $\mathcal{C} \in \mathbb{F}_2^N$ is contemplated, where $|\mathcal{C}| = 2^K$. A codeword $\boldsymbol{x} \in \mathcal{C}$ is sent across the BEC and received at the receiver. Thereafter, the received word is decoded using the message passing algorithm. So, all the received messages are only from the set $\{0, 1, \Delta\}$ (where, Δ means "erasure") because of the property of BEC (cf. section 2.2).

1. *Variable node operation*:

   If any one of the incoming messages is either 0 or 1 at the variable node, then the outgoing message will be 0 or 1 respectively because of the repetition code constraint at the variable node. It can be expounded in this way that if any one of the incoming message at the variable node is 0, then it will be sure that the local code constraint is all-zeros because of the property of the repetition code at the variable node. It can also be said that the outgoing message will be Δ, if and only if all the incoming messages are Δ. The variable node operations for the BEC are shown in the Figure 4.13.

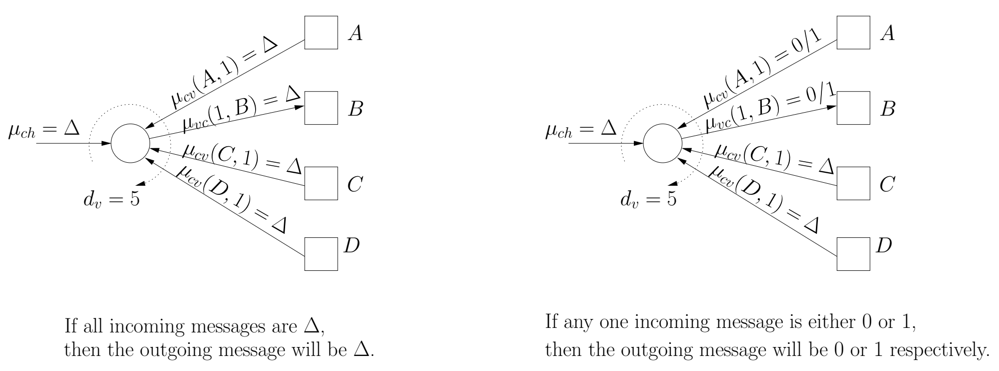


Figure 4.13: *An isolated and independent variable node operation for the BEC.*

2. *Check node operation*:

   If any one incoming message is Δ at the check node, then the outgoing message will be Δ because of the single parity check code constraint, see Figure 4.14 (i). It can be reiterated in this way that if any one of the incoming message is Δ at the check node, then the local code constraint will be in dilemma that what it should consider incoming Δ as 0 or 1 to fulfill the local code constraint which is a single parity check code. It can also be said that if the incoming messages are 0 and 1, then the outgoing message will be either 0 or 1 to fulfill the single parity check code constraint, see Figure 4.14 (ii).

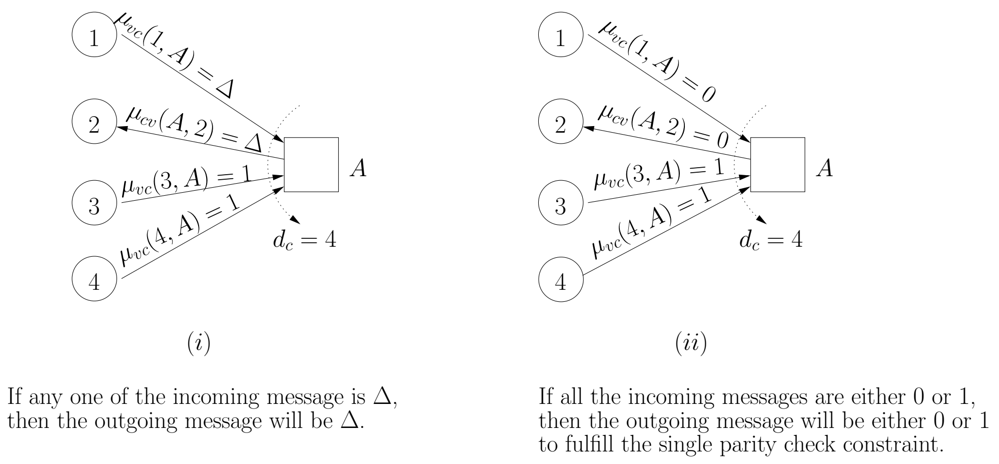

1
2
3
4
A
$\mu_{vc}(1,A) = \Delta$
$\mu_{cv}(A,2) = \Delta$
$\mu_{vc}(3,A) = 1$
$\mu_{vc}(4,A) = 1$
$d_c = 4$
(i)
If any one of the incoming message is Δ, then the outgoing message will be Δ.
$\mu_{vc}(1,A) = 0$
$\mu_{cv}(A,2) = 0$
$\mu_{vc}(3,A) = 1$
$\mu_{vc}(4,A) = 1$
$d_c = 4$
(ii)
If all the incoming messages are either 0 or 1, then the outgoing message will be either 0 or 1 to fulfill the single parity check constraint.

Figure 4.14: *An isolated and independent check node operation for the BEC.*

3. *Termination of the iteration*:

   The iteration will terminate when all the code symbols are recovered or the maximum number of iterations have reached, otherwise go to step 1. After the termination of the algorithm, if all the code symbols are not recovered, then the set of code symbols which are not resolvable will be called as a *stopping set*.

4.5.2 Stopping Sets

A stopping set is a set of the code symbols which are not resolvable by the MPA. Formally, the stopping set $\mathcal{S}$ is a subset of $\mathcal{V}$ (where, $\mathcal{V}$ is the set of all variable nodes n), such that the all neighbours of $\mathcal{S}$ i.e., $\mathcal{N}(\mathcal{S})$ have *at least two* connections to $\mathcal{S}$. It can be said that if the neighbouring check nodes $\mathcal{N}(\mathcal{S})$ have at least 2 incoming erasures from the variable nodes in the set $\mathcal{S}$, then the outgoing message μ_{cv} will always be an erasure. Thus, the erasures travel back and forth through the edges between the stopping set $\mathcal{S}$ and the neighbouring check nodes. Therefore, the stopping set $\mathcal{S}$ will have unresolvable variable nodes.

To be precise, a set $\mathcal{E}$ is considered as a subset of $\mathcal{V}$ having erasures from the channel. So, if $\mathcal{E} \supseteq \mathcal{S}$, then the set $\mathcal{S}$ will form a set of non-resolvable code symbols and it won't be possible to decode the variable nodes in the set $\mathcal{S}$ under BEC.

The $(7,4,3)$ Hamming code $\mathcal{C}$ is cogitated on to explain the concepts of a stopping set. If a codeword $\boldsymbol{x} \in \mathcal{C}$ is transmitted across BEC, then the received word will be $\boldsymbol{y}$ which is decoded by the MPA. It can be construed that the various combinations of the variable nodes can be found to form a stopping set. So, one of the stopping set $\mathcal{S} = \{1,2,3\}$ is considered and shown in the Figure 4.15 such that the $\mathcal{N}(\mathcal{S}) = A, B$ must have at least two connections to $\mathcal{S}$ by the definition of the stopping set. If the received word $\boldsymbol{y}$ is $\begin{bmatrix}\Delta & \Delta & \Delta & 0 & 0 & 0 & \Delta\end{bmatrix}$, then the erasure set will be $\mathcal{E} = \{1,2,3,7\}$ such that $\mathcal{E} \supseteq \mathcal{S}$, which have received erasures from the channel. It can easily be seen in the Figure 4.15 that the set $\mathcal{S} = \{1,2,3\}$ can never be decoded under these conditions except x_7 which is decoded by the MPA. In this figure, the different colours are used to denote the various messages. It should also be seen that the messages shown within the variable nodes are the messages recovered finally ($\hat{x}_n$) after one iteration. So after the termination of the iteration, the decoded codeword $\hat{\boldsymbol{x}}$ is $\begin{bmatrix}\Delta\ \Delta\ \Delta\ 0\ 0\ 0\ 0\end{bmatrix}$, thus the set $\mathcal{S} = \{1,2,3\}$ is a stopping set of unresolvable code symbols. Furthermore, it can easily be seen that the messages within the variable nodes remain the same after any further iterations too.

If $\mathcal{E} \subset \mathcal{S}$, then all the code symbols can be resolved. If the received word $\boldsymbol{y}$ is $\begin{bmatrix}\Delta & 0 & \Delta & 0 & 0 & 0 & 0\end{bmatrix}$, then the erasure set will be $\mathcal{E} = \{1,3\}$. The set $\mathcal{S} = \{1,2,3\}$ is considered to represent the factor graph in the Figure 4.16 of the $(7,4,3)$ Hamming code that all the code symbols are resolvable. So after the termination of the one complete iteration, the decoded codeword $\hat{\boldsymbol{x}}$ is $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$. Hence, the stopping set $\mathcal{S} = \{1,2,3\}$ is resolvable when $\mathcal{E} \subset \mathcal{S}$ otherwise it is not resolvable when $\mathcal{E} \supseteq \mathcal{S}$.

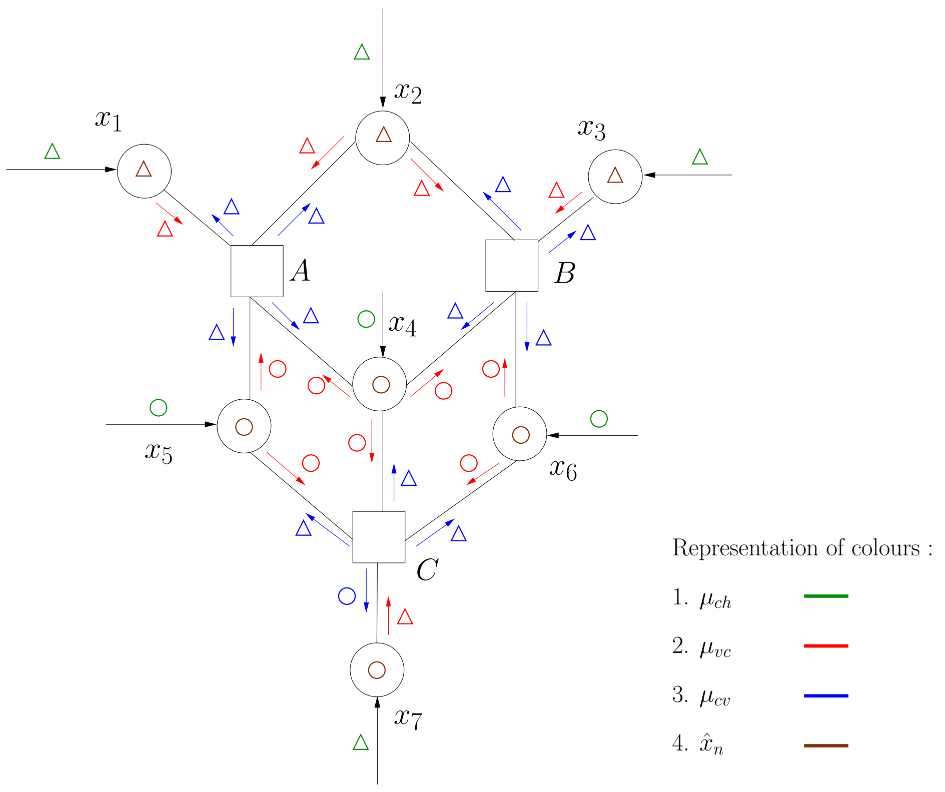


Figure 4.15: *The set $\mathcal{S} = \{1, 2, 3\}$ is shown to be a stopping set in the $(7, 4, 3)$ Hamming code.*

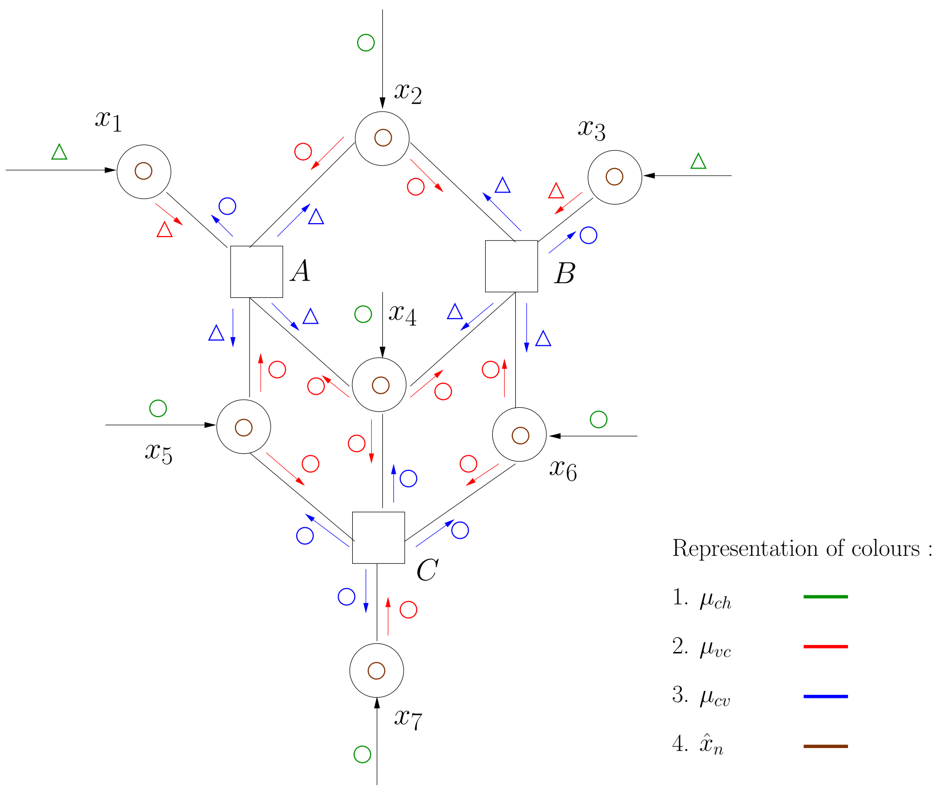


Figure 4.16: *The stopping set $\mathcal{S} = \{1, 2, 3\}$ will be resolvable, if $\mathcal{E} = \{1, 3\}$ such that $\mathcal{E} \subset \mathcal{S}$.*

Linear Programming Decoding 5

5.1 Introduction

Generally, the decoding of linear block codes, and particularly the LDPC codes can be carried out by several methods. One of the well-known method, the message passing algorithm (MPA) was studied in the previous chapter. Another method will be described in this chapter, the *linear programming* decoding.
The Linear Programming (LP) method is an optimization method to solve a problem defined by a linear *objective function* using *linear* constraints in $\mathbb{R}$.

The optimization problem can be considered as,
Given a binary linear code $\mathcal{C} \in \mathbb{F}_2^N$, a codeword $\boldsymbol{x} = [x_1\ x_2\ \dots\ x_N]^T$ is transmitted over a memoryless channel, the received vector is $\boldsymbol{y} = [y_1\ y_2\ \dots\ y_N]^T$ and the log likelihood ratio (LLR) vector is $\boldsymbol{\lambda} = [\lambda_1\ \lambda_2\ \dots\ \lambda_N]$. The parity check matrix $\underline{\mathbf{H}} \in \mathbb{F}_2^{M \times N}$. The goal of the optimization problem is to find the maximum likelihood codeword,

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{x} \in \mathcal{C} \end{aligned} \tag{5.1}$$

From now onwards the optimization problem is written with the notation "min." as minimum and "s.t." as subject to. If $\boldsymbol{x} \in \mathcal{C}$, then $\underline{\mathbf{H}} \otimes \boldsymbol{x} = 0$. So, the equation (5.1) becomes,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \underline{\mathbf{H}} \otimes \boldsymbol{x} = 0 \\ & \boldsymbol{x} \in \{0, 1\} \end{aligned} \tag{5.2}$$

The different parts of this optimization problem are,

- An *objective function* which is minimized (or sometimes maximized):

$$\text{min.} \quad \boldsymbol{\lambda}^T \boldsymbol{x} \tag{5.3}$$

- The *problem constraint*,

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = 0 \tag{5.4}$$

  If we split $\underline{\mathbf{H}}$ into its M rows, the equation (5.4) will be,

$$\boldsymbol{h}_m \otimes \boldsymbol{x} = 0, \qquad m = \underbrace{A, B, C, ...}_{M} \tag{5.5}$$

  We have now M equations, each corresponding to

$$a_1 x_1 \oplus a_2 x_2 \oplus ... \oplus a_N x_N = 0 \tag{5.6}$$

  with $a_n \in \{0, 1\}, \quad n = 1, .., N$.

- The *integer (binary) variable*,

$$x_n \in \{0, 1\} \tag{5.7}$$

The optimization problem will be described to solve through linear programming in this chapter. In section 5.2, a linear maximum likelihood function is derived in order to be used in the LP formulation which is explained in the section 5.3. After the analytical formulation of the LP decoding problem, a geometrical interpretation will be given in section 5.4. An alternative formulation of LP decoding is described in the section 5.5. In section 5.6, the notion of pseudocodewords which are specific to our problem will be described before ending with some ideas to improve the performance of LP. The end of this chapter shows some problems observed through the particular case of the binary symmetric channel in section 5.7 and an improved algorithm for the LP decoder is given in section 5.8.

5.2 Maximum Likelihood Decoding for LP

After the transmission of an error-correcting code, the receiver has to decode the transmitted codeword. So, one way to decode is to choose a codeword which has the maximum likelihood probability of a received word given a transmitted codeword. It means that it will find a *maximum likelihood codeword*. This section will show how the maximum likelihood codeword can be derived in a linear form in order to be used in the LP formulation.

If a codeword $\boldsymbol{x} \in \mathcal{C}$ is transmitted over a memoryless channel and the corresponding received vector is $\boldsymbol{y}$, then the maximum likelihood codeword can be:

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmax}} \ \Pr(\, \boldsymbol{y} \,|\, \boldsymbol{x} \,) \tag{5.8}$$

Since, the variables are independent and the channel is memoryless without feedback, equation (5.8) becomes:

$$\begin{aligned} \hat{\boldsymbol{x}} &= \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmax}} \prod_{n=1}^{N} \Pr(\, y_n \,|\, x_n \,) && (5.9) \\ &= \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \left(-\ln \prod_{n=1}^{N} \Pr(\, y_n \,|\, x_n \,) \right) && (5.10) \\ &= \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \left(-\sum_{n=1}^{N} \ln \Pr(\, y_n \,|\, x_n \,) \right) && (5.11) \end{aligned}$$

One trick is now used. The term $\sum_{n=1}^{N} \ln \Pr(\, y_n \,|\, 0 \,)$ is independent of $\boldsymbol{x}$. So, it can be considered as a constant since minimization is done over $\boldsymbol{x}$. Equation (5.11) will be,

$$\begin{aligned} \hat{\boldsymbol{x}} &= \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \sum_{n=1}^{N} \ln \Pr(\, y_n \,|\, 0 \,) - \sum_{n=1}^{N} \ln \Pr(\, y_n \,|\, x_i \,) && (5.12) \\ &= \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \sum_{n=1}^{N} \ln \frac{\Pr(\, y_n \,|\, 0 \,)}{\Pr(\, y_n \,|\, x_n \,)} && (5.13) \end{aligned}$$

The sum,

$$\sum_{n=1}^{N} \ln \frac{\Pr(\, y_n \,|\, 0 \,)}{\Pr(\, y_n \,|\, x_n \,)}$$

is equal to 0 when $x_n = 0$ and for $x_n = 1$, it is equal to,

$$\sum_{n=1}^{N} \ln \frac{\Pr(\, y_n \,|\, x_n = 0 \,)}{\Pr(\, y_n \,|\, x_n = 1 \,)} = \sum_{n=1}^{N} \lambda_n$$

So, equation (5.13) is now,

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \sum_{n=1}^{N} x_n \ \cdot \ \left(\ln \frac{\Pr(\, y_n \,|\, x_n = 0 \,)}{\Pr(\, y_n \,|\, x_n = 1 \,)} \right) \tag{5.14}$$

Finally, the log likelihood ratio can be replaced by its value such that,

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \sum_{n=1}^{N} \lambda_n x_n \tag{5.15}$$

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmin}} \, \boldsymbol{\lambda}^T \boldsymbol{x} \tag{5.16}$$

We see that the equation (5.16) is exactly equal to our optimization problem described in equation (5.1).

5.3 Linear Programming Formulation

In this section, we will first state the problem of linear programming decoding and then describe the results.

5.3.1 Problem Formulation

An example will be taken in this section to derive a formulation of LP decoding. Afterwards the general formulation is shown.

Our goal is still,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{h}_m \otimes \boldsymbol{x} = 0, \quad m = \underbrace{A, B, C, ...}_{M} \\ & x_n \in \{0, 1\} \end{aligned} \tag{5.17}$$

We want to use a linear programming decoder such that the variables and constraint function are in $\mathbb{R}$. A simple example is considered to formulate the integer LP problem before relaxation of the constraints. The *relaxation of the constraints* means that the constraint $x_n \in \{0, 1\}$ is changed to $x_n \in [0, 1]$.

The check equations in $\mathbb{F}_2$ will be reformulated as linear equations in $\mathbb{R}$ which is described further. So, consider the following parity check matrix $\underline{\mathbf{H}}$,

$$\underline{\mathbf{H}} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{5.18}$$

If $\boldsymbol{x} = [x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6]^T$, the set of check equations corresponding to the $\underline{\mathbf{H}}$ matrix will be,

$$\underline{\mathbf{H}} \otimes \boldsymbol{x} = \left\{ \begin{matrix} chk(A) : x_1 \oplus x_4 \oplus x_5 = 0 \\ chk(B) : x_2 \oplus x_4 \oplus x_6 = 0 \\ chk(C) : x_3 \oplus x_5 \oplus x_6 = 0 \end{matrix} \right\} \tag{5.19}$$

In order to use the parity check equation in $\mathbb{R}$, a new formulation with new variables is used. Let us define them for example for $chk(A)$,

- $\mathbb{S}_A$ is the set of indices of the code symbols used in the check equation $chk(A)$

$$\mathbb{S}_A = \{1, 4, 5\} \tag{5.20}$$

- $\boldsymbol{x}_{\mathbb{S}_A}$ is the local codeword formed with the corresponding code symbols used in check equation $chk(A)$:

$$\boldsymbol{x}_{\mathbb{S}_A} = (x_1, x_4, x_5)^T \tag{5.21}$$

- The matrix which will be able to extract $\boldsymbol{x}_{\mathbb{S}_A}$ from $\boldsymbol{x}$ is called $\underline{\mathbf{B}}_A$

$$\underline{\mathbf{B}}_A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.22}$$

It can be verified that

$$\boldsymbol{x}_{\mathbb{S}_A} = \underline{\mathbf{B}}_A \boldsymbol{x} \tag{5.23}$$

Notice that in equation (5.23), the addition and multiplication in $\mathbb{R}$ are used instead of the modulo 2 operation.

- Now, another matrix $\underline{\mathbf{A}}_A$ is defined as follows.

The columns of $\underline{\mathbf{A}}_A$ are composed of the local codewords satisfying $chk(A)$. These local codewords are even weight vectors of length d_c, including the zero vector. In our case $d_c = 3$, so:

$$\boldsymbol{x}_{\mathbb{S}_A} \in \{(0,0,0)^T; (0,1,1)^T; (1,0,1)^T; (1,1,0)^T\}$$

Thus,

$$\underline{\mathbf{A}}_A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{5.24}$$

- An indicator vector $\boldsymbol{w}_A$ that select the right configuration for $chk(A)$ is defined in this way:

$$\boldsymbol{w}_A = \begin{bmatrix} w_{A,\{\emptyset\}} \\ w_{A,\{4,5\}} \\ w_{A,\{1,5\}} \\ w_{A,\{1,4\}} \end{bmatrix} \in \mathbb{F}_2^4, \quad \mathbb{1}^T \boldsymbol{w}_A = 1 \tag{5.25}$$

It shall be noted that the 4 in $\mathbb{F}_2^4$ corresponds to the number of different local codewords satisfying $chk(A)$ and also to the number of columns in the matrix $\underline{\mathbf{A}}_A$.

And $\mathbb{1}$ is the all-ones vector,

$$\begin{bmatrix}1\\1\\1\\1\end{bmatrix} \tag{5.26}$$

So, $\boldsymbol{w}_A$ belongs to the set because of the definition in equation 5.25,

$$\boldsymbol{w}_A \in \left\{ \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \right\} \tag{5.27}$$

All these vectors will satisfy, $\mathbb{1}^T \boldsymbol{w}_A = 1$.

It can be seen that

$$\boldsymbol{x}_{\mathbb{S}_A} = \underline{\mathbf{A}}_A \boldsymbol{w}_A \tag{5.28}$$

In the equation (5.28), the multiplication and addition are used in $\mathbb{R}$. If for example $\boldsymbol{w}_A = [0\ 1\ \ 0\ 0]^T$, then $w_{A,\{4,5\}} = 1$ and $w_{A,\{\emptyset\}} = w_{A,\{1,5\}} = w_{A,\{1,4\}} = 0$ such that $\boldsymbol{x}_{\mathbb{S}_A} = [0\ 1\ 1]^T$. $\boldsymbol{w}_A$ is an indicator/auxiliary variable which selects the correct configuration for the local codeword from the columns of $\underline{\mathbf{A}}_A$.

The equations (5.23) and (5.28) are combined such that,

$$\underline{\mathbf{B}}_A \boldsymbol{x} = \underline{\mathbf{A}}_A \boldsymbol{w}_A \qquad \boldsymbol{w}_A \in \mathbb{F}_2^4, \quad \mathbb{1}^T \boldsymbol{w}_A = 1 \tag{5.29}$$

Equation (5.29) has been formed in such a way that it satisfies the check equation $chk(A)$. Thus, equation (5.29) is equivalent to $chk(A)$.

For each check equation of (5.19), the following settings for the new linear equations can be deduced,

$$chk(A): \left\{ \begin{array}{l} x_1 \oplus x_4 \oplus x_5 = 0, \qquad \mathbb{S}_A = \{1,4,5\} \\ \\ \boldsymbol{w}_A = \begin{bmatrix} w_{A,\{\emptyset\}} & w_{A,\{4,5\}} & w_{A,\{1,5\}} & w_{A,\{1,4\}} \end{bmatrix}^T \\ \\ \underline{\mathbf{B}}_A = \begin{bmatrix} 1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0 \end{bmatrix}, \quad \underline{\mathbf{A}}_A = \begin{bmatrix} 0&0&1&1\\0&1&0&1\\0&1&1&0 \end{bmatrix} \end{array} \right\} \tag{5.30}$$

$$chk(B) : \left\{ \begin{array}{l} x_2 \oplus x_4 \oplus x_6 = 0, \qquad \mathbb{S}_B = \{2,4,6\} \\ \\ \boldsymbol{w}_B = \left[\, w_{B,\{\emptyset\}}\ w_{B,\{4,6\}}\ w_{B,\{2,6\}}\ w_{B,\{2,4\}} \right]^T \\ \\ \underline{\mathbf{B}}_B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \underline{\mathbf{A}}_B = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \right\} \tag{5.31}$$

$$chk(C) : \left\{ \begin{array}{l} x_3 \oplus x_5 \oplus x_6 = 0, \qquad \mathbb{S}_C = \{3,5,6\} \\ \\ \boldsymbol{w}_C = \left[\, w_{C,\{\emptyset\}}\ w_{C,\{5,6\}}\ w_{C,\{3,6\}}\ w_{C,\{3,5\}} \right]^T \\ \\ \underline{\mathbf{B}}_C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \underline{\mathbf{A}}_C = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \right\} \tag{5.32}$$

So, if the number of ones is the same in each row of $\underline{\mathbf{H}}$ ($d_c = 3$), $\underline{\mathbf{A}}_m$ will be the same matrix for each check equation (m corresponds to the index of the check equation). Let us generalize the formulation. For each m^{th} row of the parity check matrix, let $d_c(m)$ be the number of ones in this row, $\mathbb{S}_m$ be the set of indices of the code symbols participating in the check equation m, $\underline{\mathbf{B}}_m$ be a $d_c(m) \times N$ matrix which extracts the local codeword $\boldsymbol{x}_{\mathbb{S}_A}$ from the codeword $\boldsymbol{x}$ and $\underline{\mathbf{A}}_m$ be a matrix $d_c(m) \times 2^{d_c(m)-1}$ whose columns are formed by the local codewords satisfying m^{th} check equation. The m^{th} parity check equation can be written as,

$$\underline{\mathbf{B}}_m \boldsymbol{x} = \underline{\mathbf{A}}_m \boldsymbol{w}_m \qquad \boldsymbol{w}_m \in \{0,1\}^{2^{d_c(m)-1}}, \quad \mathbb{1}^T \boldsymbol{w}_m = 1 \tag{5.33}$$

It can be noticed that all the operations used in this equation are over $\mathbb{R}$. By replacing the check constraint in (5.17) by this new equation (5.33), the optimization formulation becomes:

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \qquad (5.34) \\ \text{s.t.} \quad & \underline{\mathbf{B}}_m \boldsymbol{x} = \underline{\mathbf{A}}_m \boldsymbol{w}_m, \quad m = \underbrace{A, B, C, ...}_{M} \\ & \boldsymbol{w}_m \in \{0,1\}^{2^{d_c(m)-1}}, \quad \mathbb{1}^T \boldsymbol{w}_m = 1, \\ & x_n \in \{0,1\}, \quad n = 1, .., N \end{aligned}$$

Now, all the constraints are linear operations over $\mathbb{R}$ but we are in the case of *Integer Linear Programming (ILP)* because x_n and w_m are integer. ILP can be NP-hard in practical situations i.e., in worst case undecidable [19] [22]. In this project, we are not interested in using ILP but linear programming decoding. So, the relaxation of the constraints are done. w_m and x_n are taken in $[0, 1]$ instead of $\{0, 1\}$ such that they can have values between 0 and 1. The ILP decoder in equation (5.34) will be changed to relaxed LP decoder which is shown in the equation 5.35,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \underline{\mathbf{B}}_m \boldsymbol{x} = \underline{\mathbf{A}}_m \boldsymbol{w}_m, \quad m = \underbrace{A, B, C, ...}_{M} \\ & \boldsymbol{w}_m \in [0,1]^{2^{d_c(m)-1}}, \quad \mathbb{1}^T \boldsymbol{w}_m = 1, \\ & x_n \in [0,1], \quad n = 1, .., N \end{aligned} \tag{5.35}$$

5.3.2 Solution of LP

After the formulation of the problem, the result given by a LP decoder could be known. And of course, we are looking for a successful decoding. In other words, the codeword decoded should exactly be the maximum likelihood codeword.

The solution of the LP problem is not always a binary codeword (we talk about integer solution) because of the relaxation of the problem. The LP decoder gives *non-integer or integer solution*.

If a solution is a non-integer solution, the LP decoder will fail. The non-integer solution is also called as a *fractional solution* and will be studied later in section 5.4.3. Fortunately, some methods like the addition of redundant parity check equations are studied in section 5.8 which may change the fractional solution to integer solution. So, an LP decoder can be improved to render an integer solution or a ML codeword after some iterations by addition of redundant parity check cuts.

When the solution is integer, it is not exactly sure to be a real transmitted codeword, but it is sure to be a ML codeword. Thus, the property of a linear programming decoder is *ML certificate*. The ML certificate is one of the greatest advantage of LP. It is said that if the LP decoding algorithm outputs a codeword (integer solution), it will be guaranteed to be a ML codeword [4]. It shall be accentuated that the sum-product algorithm (SPA) may give a non codeword binary vector; however the LP decoder will always give a ML codeword, if the solution is integer [4]. This property can be demonstrated as follows,

Consider an integer solution output by the solver, $\hat{\boldsymbol{x}}$ and $\boldsymbol{w}$. Our LP formulation states that $\mathbb{1}^T \boldsymbol{w}_m = 1$. So, when $\boldsymbol{w}$ is integer all the bits of $\boldsymbol{w}_m$ are 0 except one bit which is 1. Thus, according to equation (5.28), $\hat{\boldsymbol{x}}_{\mathbb{S}_m}$ will be a local codeword. Hence, from equation (5.23), $\hat{\boldsymbol{x}}$ is a codeword. And if we remind the following definition of ML decoding:

$$\hat{\boldsymbol{x}} = \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{C}} \boldsymbol{\lambda}^T \boldsymbol{x} \tag{5.36}$$

By definition, $\hat{\boldsymbol{x}}$ is then the ML codeword. The ML certificate is really important in the field of decoding and allows a great attraction for LP decoding. Another important theorem of LP decoding is that it is optimal (ML decoding) for cycle free graphs. We did not prove it in the project but it is referred in [4][5]. Note that the algorithm of LP decoding has been implemented during the project and can be seen in the Matlab code *lclp_example.m* or *lclp.m* in the CD-ROM provided with the report.

5.3.3 Scaling of $\boldsymbol{\lambda}$ (noise)

Indeed, the LP formulation consists in minimizing the linear cost function: $\operatorname{argmin} \boldsymbol{\lambda}^T \boldsymbol{x}$. This means that if a factor β is added, the estimation will still be same,

$$\hat{\boldsymbol{x}} = \operatorname{argmin} \boldsymbol{\lambda}^T \boldsymbol{x} = \operatorname{argmin}(\beta \cdot \boldsymbol{\lambda}^T) \boldsymbol{x} \tag{5.37}$$

As $\boldsymbol{\lambda}$ can be scaled by a constant β, then the knowledge of the noise is not important (cf. section 2.5).

5.4 Geometric Interpretation

In this section some geometric interpretations of the linear programming formulation will be shown.

5.4.1 The Local Codeword Constraint gives a Convex Hull

Consider now the constraints for check node A in the example, shown in equation 5.38 for convenience.

$$\begin{bmatrix} x_1 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_{A,\{\emptyset\}} \\ w_{A,\{1,4\}} \\ w_{A,\{1,5\}} \\ w_{A,\{4,5\}} \end{bmatrix} \quad , \quad w_{A,S} \geq 0 \quad , \quad \mathbb{1}^T \boldsymbol{w} = 1 \tag{5.38}$$

Where, $\boldsymbol{w}_A$ has been written out, such that for example $w_{A,\{1,4\}}$ is the indicator auxiliary variable selecting $x_1 = x_4 = 1$ and $x_5 = 0$. Consider now the definition of the convex hull of a set $\mathcal{L}$.

$$\text{conv}(\mathcal{L}) = \left\{ \sum_{\boldsymbol{l}\in\mathcal{L}} \gamma_l \, \boldsymbol{l} : \ \gamma_l \geq 0 \, , \ \sum_{\boldsymbol{l}\in\mathcal{L}} \gamma_l = 1 \right\} \tag{5.39}$$

The *convex hull* of a set, is all the points that are a linear combinations of points in the set, where all weights are non-negative and sum to one. The constraints in (5.38) on w are equivalent to those for the convex hull γ in (5.39). Defining the matrix

$$\underline{\mathbf{A}}_A = [\boldsymbol{p}_1 \ \boldsymbol{p}_2 \ \boldsymbol{p}_3 \ \boldsymbol{p}_4] \tag{5.40}$$

such that the set $E_A = \{\boldsymbol{p}_i : i = 1, 2 \ldots 2^{(d_c-1)}\}$, are the columns of $\underline{\mathbf{A}}_A$. When describing the convex hull, the elastic band analogy is often used [20], where a rubber band is released around the points in the set. The rubber band is then the surface of the solid figure that the convex hull describes. The Figure 5.1 shows the convex hull of E_A, a solid *tetrahedron*.

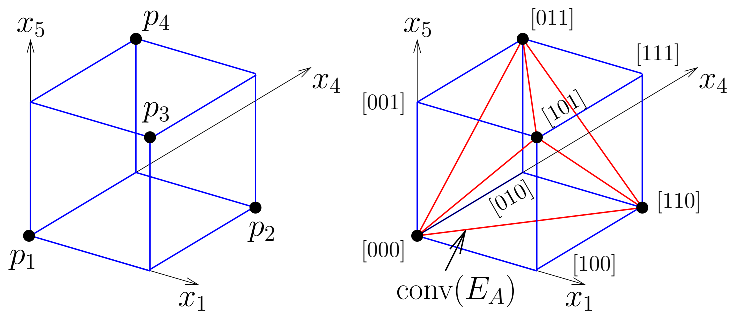


Figure 5.1: *Points $\mathbf{p}_i$ of the set E_A to the left is applied with a rubber band which render a solid tetrahedron shown in the figure to the right.*

This means that $\boldsymbol{x}_{\mathbb{S}_A} \in \text{conv}(E_A)$, i.e. all the points in the tetrahedron. For a weight 3 ($d_c = 3$) check equation, the solution space will be a tetrahedron in 3 dimensions. For higher degree it will be *polytopes* in multiple dimensions. Each check equation A, B, C in the example has a solution space as a tetrahedron as shown in the Figure 5.2.

In the Figure 5.1, only constraints on x_1, x_4, x_5 are considered whereas the rest, x_2, x_3, x_6, are free in check equation A. If it's written as an optimization problem, it can be formulated as,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \text{conv}(\mathcal{C}_m) \qquad \forall \quad m = \underbrace{A, B, C \ldots}_{M} \end{aligned} \tag{5.41}$$

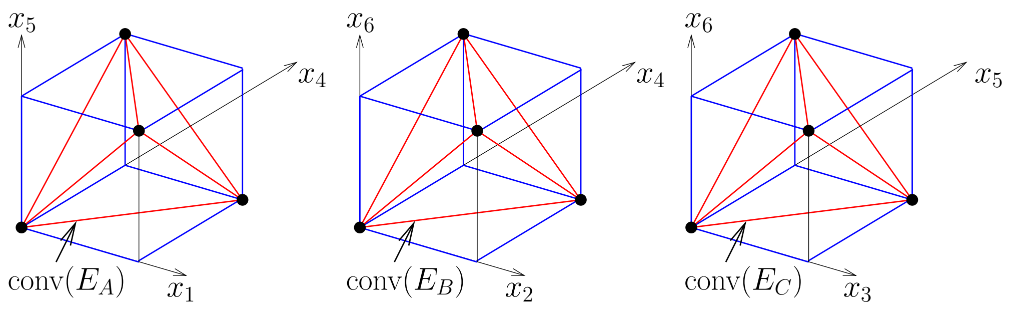


Figure 5.2: *The three tetrahedrons of the example considered in LP chapter.*

We are considering the intersection of our solution space, which means that the total solution space for our LP relaxation will be the polytope P, in general.

$$P = \text{conv}(\mathcal{C}_A) \cap \text{conv}(\mathcal{C}_B) \cap \text{conv}(\mathcal{C}_C) \cdots \tag{5.42}$$

Note the similarity between (5.42) and the definition $\mathcal{C} = \mathcal{C}_A \cap \mathcal{C}_B \cap \mathcal{C}_C$. Equation (5.41) can easily be rewritten as,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in P \end{aligned} \tag{5.43}$$

5.4.2 Possible Solutions

One question arises. Have we not introduced an infinite number of solution to our optimization problem by considering the convex hull instead of just the vertices? The solution space is not just a set of vertices but a solid polytope. This problem occurred when we considered the relaxations in the interval $x_n \in [0,1]$ instead of the binary values $x_n \in \{0,1\}$. Consider now the following theorem,

A basic feasible solution is an extreme point to the linear program $\underline{\mathbf{A}}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{x} \geq 0, \quad \mathbf{c}^T\boldsymbol{x}$ *is minimum* [3, p.17]

This means that $x^* \in V(P)$, i.e., the solution will always be a *vertex* (extreme point) of the polytope P (the solution space is bounded). So, even though the constraint space is large, optimization will only yield the vertices of the constrained space. This means that the following two optimization problems will exactly give the same solution considering our results from the definition of the convex hull in section 5.4.1.

$$\underset{\boldsymbol{x} \in \mathcal{C}}{\text{argmin}}\, \boldsymbol{\lambda}^T \boldsymbol{x} = \underset{\boldsymbol{x} \in \text{conv}(\mathcal{C})}{\text{argmin}}\, \boldsymbol{\lambda}^T \boldsymbol{x} \tag{5.44}$$

So, if we just are considering the convex hull of the code $\mathcal{C}$, the solution will be the (optimal) ML solution. But, in general, we are solving the problem.

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in P \end{aligned}$$

Where, $P \supseteq \text{conv}(\mathcal{C})$[4]. The polytope P is larger than conv($\mathcal{C}$). This means that all codewords (integer solutions) are vertices of the polytope P along with some other solutions which are called fractional solutions. The reason for this is that [4, p.61].

$$\mathcal{C} = P \cap \{0,1\}^N \tag{5.45}$$

The set of vertices having integer points in the polytope P is the code C. The rest vertices having non-integer solutions are fractional solutions. Equation (5.45) says that the polytope is "proper". It was exactly this property that proved the ML certificate in section 5.3.2, just written in term of the polytope P.

5.4.3 Description of the Polytope P

In this section the previous definitions will be described in a more intuitive and logical way. Consider the Figure 5.3, where the polytope P is described as a 2 dimensional figure.

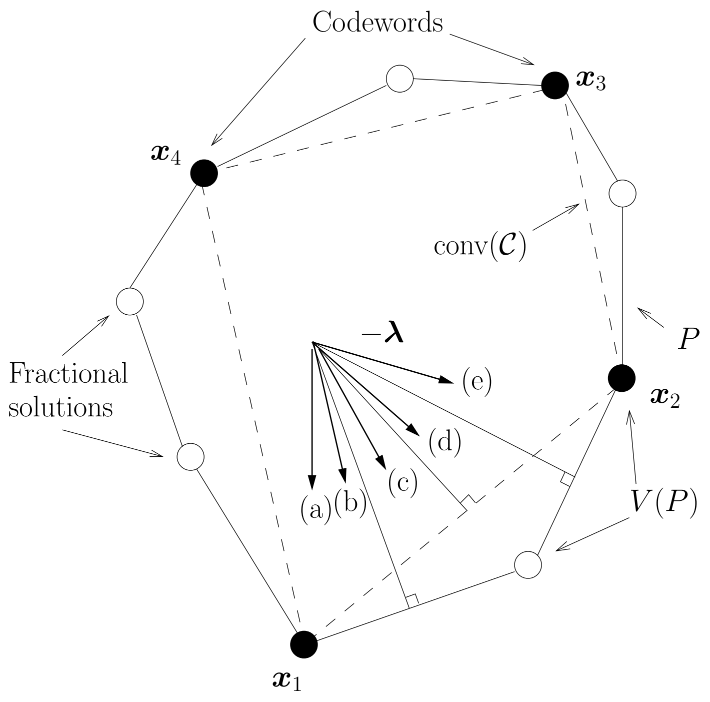


Figure 5.3: *Description of the relation between the polytope P, conv($\mathcal{C}$), codewords, fractional solutions and LLR vector $\boldsymbol{\lambda}$. Idea from [4, p.43].*

Figure 5.3 shows the polytope P and all the vertices as circles ($\bullet/\circ$). It is clear that $\text{conv}(\mathcal{C})$ is a subset of P. The optimization problem is described in an intuitive way as described in [4, p.44]. Turn the polytope such that $-\boldsymbol{\lambda}$ points in the direction of gravity. Then drop a ball within the polytope. The vertex where the ball settles down is the optimal solution. Observe now the outcome of the different received LLR vector $\boldsymbol{\lambda}$ given that the codeword $\boldsymbol{x}_1$ was transmitted. The cases (a)-(e) corresponds to increased amount of noise added to the transmitted codeword.

(a) No noise. Both ML and LP decoder succeed.

(b) Both ML and LP decoder succeed.

(c) LP decoder will give a fractional solution. LP fails, but it will be detected because it renders a non-integer solution (fractional solution). ML decoder succeeds.

(d) Both ML and LP decoder fails. But for the LP decoder, the failure is detected because the solution is non-integer.

(e) Both ML and LP decoder fails. Failure is not detected for LP because it renders an integer (codeword) solution.

Under a MPAM scheme we can also understand the cases (a)-(e) as in the Figure 5.4, where the transmitted vector $\boldsymbol{x}_1'$ is added with ”Gaussian distributed” noise.

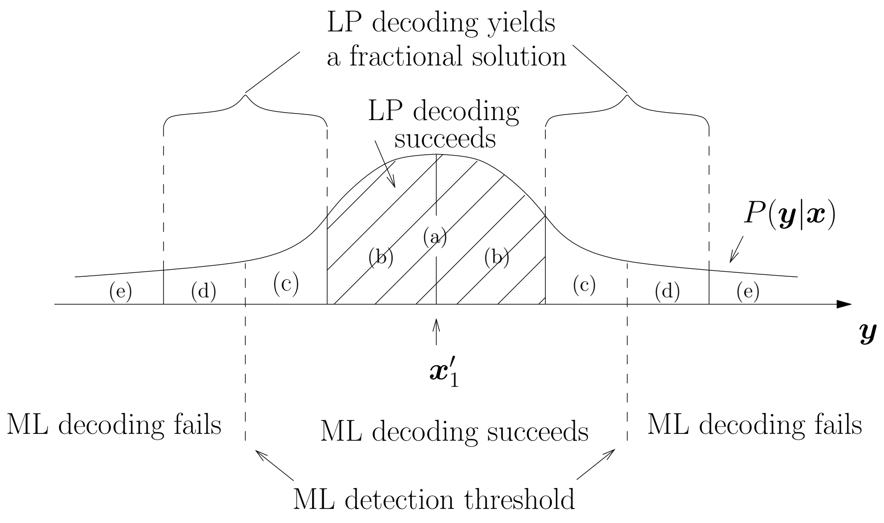


Figure 5.4: The different cases (a)-(e) under a MPAM scheme. Giving meaning to the increased noise from the cases (a) to (e).

5.5 Alternative Formulation

In this section, an alternative way of defining the constraints of the LP are given. The motivation for the *alternative formulation* is to reduce the complexity of LP by removing the auxiliary variable $w_{m,S}$. We know that for each check equation, the number of different auxiliary variables $w_{m,S}$ grows exponentially (2^{d_c-1}), so removing these could simplify the problem significantly. However, we are minimizing $\boldsymbol{\lambda}^T \boldsymbol{x}$ and the objective function is independent of the $w_{m,S}$, so it can be possible to remove the auxiliary variable.

5.5.1 Exemplification of the Alternative Formulation

In this section, an example is given to show how the alternative formulation can be derived. Consider the constraints for the degree 3 check equation A in equation (5.46), where the constraint $\mathbb{1}^T \boldsymbol{w}_A = 1$ has been added to the matrix.

$$\begin{bmatrix} 1 \\ x_1 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_{A,\{\emptyset\}} \\ w_{A,\{1,4\}} \\ w_{A,\{1,5\}} \\ w_{A,\{4,5\}} \end{bmatrix} \quad , \quad w_{A,S} \geq 0 \tag{5.46}$$

Formulating this in an augmented matrix form and bringing it into row reduced echelon form.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & x_1 \\ 0 & 1 & 0 & 1 & x_4 \\ 0 & 0 & 1 & 1 & x_5 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{2-x_1-x_4-x_5}{2} \\ 0 & 1 & 0 & 0 & \frac{x_1+x_4-x_5}{2} \\ 0 & 0 & 1 & 0 & \frac{x_1-x_4+x_5}{2} \\ 0 & 0 & 0 & 1 & \frac{-x_1+x_4+x_5}{2} \end{bmatrix} \tag{5.47}$$

If we add add the constraint that $w_{A,S} \geq 0$ and multiply with -1, we obtain the following 4 equations in (5.48).

$$\begin{aligned} -2 + x_1 + x_4 + x_5 &\leq 0 \qquad (hs1) \\ -x_1 - x_4 + x_5 &\leq 0 \qquad (hs2) \\ -x_1 + x_4 - x_5 &\leq 0 \qquad (hs3) \\ +x_1 - x_4 - x_5 &\geq 0 \qquad (hs4) \end{aligned} \tag{5.48}$$

First note that the solution space of x_n in (5.48) should be equivalent with the solution space of x_n in (5.46) (all the constraints are utilized). We can now describe the local check equation constraints in another way without the indicator variable $w_{A,S}$. Further, the new constraints have a nice interpretation. The equation (5.48) describes 4 *half-spaces* $(hs1) - (hs4)$. Contemplate now the tetrahedron in the Figure 5.5.

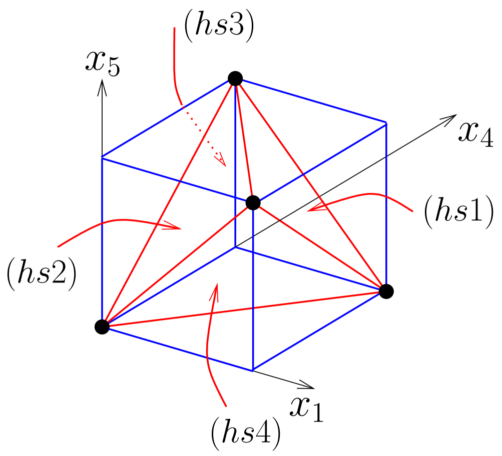


Figure 5.5: *The 4 half-spaces defining the tetrahedron.*

For equality in the equations (5.48), the faces of the tetrahedron is described by the 4 planes, and the solid tetrahedron is described by the 4 half-spaces $(hs1) - (hs4)$.

5.5.2 The Alternative Formulation in General

From the previous section's results, it can be construed that the convex hull can be described as a set of inequalities without using the auxiliary variable $w_{m,S}$. Firstly, the example of the previous section with half space $(hs2)$ is described after which the alternative formulation is derived in general. In the Figure 5.6, the example with $(hs2)$ is shown.

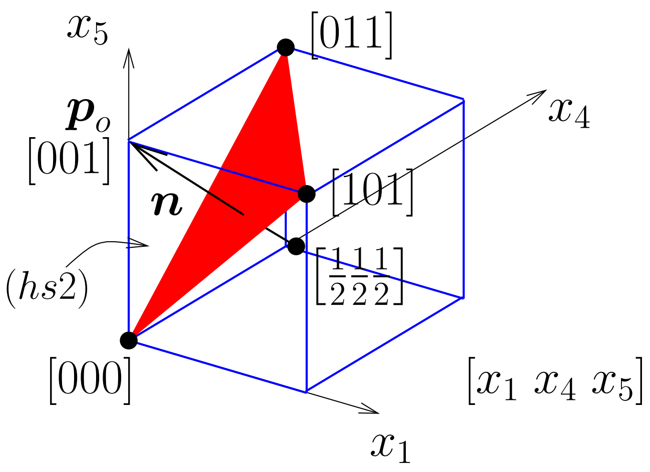


Figure 5.6: *Half-space 2 $(hs2)$ with the normal $\boldsymbol{n}$.*

The normal to the half-space is $\boldsymbol{n} = \boldsymbol{p}_o - \frac{1}{2}\mathbb{1}$, i.e. the vector from the center of the unit cube to an odd weighted vertex $\boldsymbol{p}_o$. Then, the plane can be defined such that all $\boldsymbol{x}_{\mathbb{S}_m}$ satisfy equation (5.49).

$$\boldsymbol{n}^T(\boldsymbol{x}_{\mathbb{S}_m} - \boldsymbol{x}_{0,\mathbb{S}_m}) = 0 \tag{5.49}$$

where $\boldsymbol{n}$ is the normal, and $\boldsymbol{x}_{0,\mathbb{S}_m}$ is a point on the plane. Inserting the example from $(hs2)$ yields.

$$\begin{aligned}
2 \cdot \boldsymbol{n}^T(\boldsymbol{x}_{\mathbb{S}_m} - \boldsymbol{x}_{0,\mathbb{S}_m}) &= 2 \cdot (\boldsymbol{p}_o - \frac{1}{2}\mathbb{1})^T(\boldsymbol{x}_{\mathbb{S}_m} - \boldsymbol{x}_{0,\mathbb{S}_m}) = 0 && (5.50)\\
&= 2 \cdot \left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \right)^T \left(\begin{bmatrix} x_1 \\ x_4 \\ x_5 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = 0 && (5.51)\\
&= \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_4 \\ x_5 \end{bmatrix} = 0 && (5.52)\\
&= -x_1 - x_4 + x_5 = 0 && (5.53)
\end{aligned}$$

Not surprisingly, we have obtained the surface corresponding to half space $(hs2)$ in equation (5.48). From this example, we now generalize this procedure for the half-spaces.

$$\begin{aligned}
\boldsymbol{n}^T(\boldsymbol{x}_{\mathbb{S}_m} - \boldsymbol{x}_{0,\mathbb{S}_m}) &\le 0 && (5.54)\\
2 \cdot \boldsymbol{n}^T \boldsymbol{x}_{\mathbb{S}_m} &\le 2 \cdot \boldsymbol{n}^T \boldsymbol{x}_{0,\mathbb{S}_m} && (5.55)
\end{aligned}$$

As $\boldsymbol{n}$ always point towards an odd weighted vertex, it will have the following form.

$$2\boldsymbol{n}^T = \left[\overbrace{\underbrace{+1 +1 \ \cdot\ \cdot\ \cdot\ +1}_{V,\ |V| \text{is odd}} \underbrace{-1 -1 \ \cdot\ \cdot\ \cdot -1}_{\mathcal{N}(m)\backslash V}}^{\mathcal{N}(m)} \right] \tag{5.56}$$

$2\boldsymbol{n}^T$ can be permuted and just shown in this regular form for convenience. V is defined as a subset of indices from $\mathcal{N}(m)$ where $\boldsymbol{n}^T$ has positive signs. The negative signs (the rest) are then $\mathcal{N}(m)$ excluding V. We can further see that we can select $\boldsymbol{x}_{0,\mathbb{S}_m}$ as one of the even weighted vertices defining the plane. The number of positions where the symbols in $\boldsymbol{x}_{0,\mathbb{S}_m}$ and $\boldsymbol{p}_o$ differs is 1 i.e., Hamming weight or distance $w_H(\boldsymbol{x}_{0,\mathbb{S}_m} \oplus \boldsymbol{p}_o) = 1$. This fact can be construed by observing the Figure 5.6. Any walk from an even weight vertex along an edge in the hyper cube will give an odd weight vertex, and vice versa. The total weight change from an odd to an even vertex is ± 1, such that the sum of $\boldsymbol{x}_{0,\mathbb{S}_m}$ is $\mathbb{1}^T \boldsymbol{x}_{0,\mathbb{S}_m} = |V| \pm 1$.

$$\boldsymbol{x}_{0,\mathbb{S}_m} = \left[\underbrace{\overbrace{11 \;\cdots\; 1 \, 0\,0 \;\cdots\; 0}^{\mathcal{N}(m)}}_{\text{\# even } = |V| \pm 1} \right] \tag{5.57}$$

$\boldsymbol{x}_{0,\mathbb{S}_m}$ can be permuted and just shown in this regular form for convenience. It can be comprehended that $2\boldsymbol{n}^T \boldsymbol{x}_{0,\mathbb{S}_m} = |V| - 1$ from the contention. Continuing from equation (5.55).

$$2\boldsymbol{n}^T \boldsymbol{x}_{\mathbb{S}_m} \leq 2\boldsymbol{n}^T \boldsymbol{x}_{0,\mathbb{S}_m} \tag{5.58}$$

$$\sum_{n \in V} x_n - \sum_{n \in \mathcal{N}(m) \setminus V} \leq |V| - 1 \tag{5.59}$$

Since we want a set of inequalities for all the surfaces, we need to find the respective half-spaces for each odd weighted vertex in the unit cube, which corresponds to the all odd combinations of the subset V taken from $\mathcal{N}(m)$. All the solutions should be within the unit cube, so the general check constraint in the alternative formulation is given by [5] [13].

$$\sum_{n \in V} x_n - \sum_{n \in \mathcal{N}(m) \setminus V} x_n \leq |V| - 1 \qquad \forall \quad V \subseteq \mathcal{N}(m), \quad |V| \text{ odd} \qquad 0 \leq x_n \leq 1 \tag{5.60}$$

Equation (5.60) is bit hard to interpret, so if these inequalities are written in matrix form then they have a nice structure. The constraints from the degree 4 check equation A of the $(7, 4, 3)$ Hamming code is considered such that $\mathcal{N}(A) = \{1, 2, 4, 5\}$ (cf. the Matlab code *lclpa.m* or *lclpa_example.m* in the CD-ROM for the implementation of the alternative formulation).

$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_4 \\ x_5 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \quad \begin{matrix} V = \{1\} \\ V = \{2\} \\ V = \{4\} \\ V = \{5\} \\ V = \{1, 2, 4\} \\ V = \{1, 2, 5\} \\ V = \{1, 4, 5\} \\ V = \{2, 4, 5\} \end{matrix} \tag{5.61}$$

Note that the matrix has a nice diagonal symmetry with the signs when the rows are in the order shown.

5.5.3 Special Properties for a Degree 3 Check Equation

Now turn back to the degree 3 check equation, which turns out to have special properties. Consider now a different way of deriving the alternative formulation.

$$\begin{bmatrix} 1 \\ x_1 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_{A,\{\emptyset\}} \\ w_{A,\{1,4\}} \\ w_{A,\{1,5\}} \\ w_{A,\{4,5\}} \end{bmatrix} \quad , \quad w_{A,S} \geq 0 \tag{5.62}$$

$$\begin{bmatrix} 1 \\ \boldsymbol{x}_{\mathbb{S}_A} \end{bmatrix} = \underline{\mathbf{A}}_{A,e}\boldsymbol{w} \tag{5.63}$$

where, $\underline{\mathbf{A}}_{A,e}$ is the extended (e) version of the $\underline{\mathbf{A}}_A$ with $\mathbb{1}^T\boldsymbol{w}$ added in the first row. $\underline{\mathbf{A}}_{A,e}$ is non-singular and square, so calculating the inverse yields.

$$2\boldsymbol{w} = 2\underline{\mathbf{A}}_{A,e}^{-1} \begin{bmatrix} 1 \\ \boldsymbol{x}_{\mathbb{S}_A} \end{bmatrix} = \begin{bmatrix} 2 & -1 & -1 & -1 \\ 0 & 1 & 1 & -1 \\ 0 & 1 & -1 & 1 \\ 0 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{x}_{\mathbb{S}_A} \end{bmatrix} \tag{5.64}$$

The factor 2 is used to bring the matrix entries to integer values. Again using the side constraint $w_{m,S} \geq 0$ and multiply by -1,

$$\begin{bmatrix} -2 & 1 & 1 & 1 \\ 0 & -1 & -1 & 1 \\ 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{x}_{\mathbb{S}_A} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{matrix} (hs1) \\ (hs2) \\ (hs3) \\ (hs4) \end{matrix} \tag{5.65}$$

which will give us the four half spaces. This manoeuvre can only be accomplished for a degree 3 check equation, because only this matrix is square. For higher degrees more unknown occurs since $\underline{\mathbf{A}}_{A,e}$ has dimension $(d_c + 1) \times (2^{d_c - 1})$. Further, a check equation of degree 3 has a special property. To see this, add now the equations corresponding to the two half-spaces $(hs2)$ and $(hs3)$.

$$\underbrace{(-x_1 - x_4 + x_5)}_{(hs2)} + \underbrace{(-x_1 + x_4 - x_5)}_{(hs3)} \leq 0 + 0 \quad \Leftrightarrow \quad x_1 \geq 0 \tag{5.66}$$

Further by adding $(hs1)$ and $(hs4)$.

$$\underbrace{(-2 + x_1 + x_4 + x_5)}_{(hs1)} + \underbrace{(x_1 - x_4 - x_5)}_{(hs4)} \leq 0 + 0 \quad \Leftrightarrow \quad x_1 \leq 1 \tag{5.67}$$

This means that the bounding $0 \leq x_1 \leq 1$ of the variables is implied in the equations. It is also implied for x_4 and x_5. If one observes the tetrahedron in the Figure 5.5 and imagine the four planes, it is possible to see that no solutions are outside the unit cube $[0,1]^3$. That is, all the variables in a degree 3 check equation are bounded automatically because it is implied in the definition. The side constraints $0 \leq x_n \leq 1$ are not needed if x_n participates in a check equation of degree 3.

The consequence of this observation is that the complexity of LP decoding may be reduced. If all variable x_n participates in a check equation of degree 3, then no bounding is needed. The complexity in terms of the number of constraints can be reduced by $2N$ since there are 2 inequalities for each variable and N variables.

5.6 Pseudocodewords and Decoding in the BEC

Pseudocodeword is an important concept in LP decoding and vital when working with the relation to message passing algorithms. We will especially use the definition of pseudocodewords to show that decoding using MPA and LP is equivalent in the BEC. A strong indication of how well we can expect the performance of LP to be.

5.6.1 Pseudocodewords

Consider the following example, which will show the definition of pseudocodewords. For check equation A in the degree 3 example.

$$\begin{bmatrix} h_1 \\ h_4 \\ h_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_{A,\{1,4\}} \\ u_{A,\{1,5\}} \\ u_{A,\{4,5\}} \end{bmatrix} \qquad u_{A,S}, h_n \in \mathbb{Z}^+ \text{ (non-negative integers)} \tag{5.68}$$

The notation of pseudocodewords has much similarity to the local codeword constraint. The difference is that the all-zero local codeword setting is not considered (the first column of the matrix is not all-zero) and the only constraint on the variables are that they are non-negative integers. Nothing about that the auxiliary variables should sum to one. By considering all check equations, the vector $\boldsymbol{h} = [h_1 \ h_2 \ . \ . \ . \ h_N]^T$ is defined as a pseudocodeword.

Look now upon the relation between pseudocodewords and our previous definition of constraint for a local codeword. We know that our polyhedron is rational, as all entries are rational, which mean that all vertices and thereby solutions are rational [16].

This means that we can find an integer β such that

$$h_n = \beta x_n = \beta \frac{k_n}{l_n} \tag{5.69}$$

h_n is a non-negative integer. This comes immediately from the definition of rational numbers since both k_n and l_n are integers. Further, we can find the same β such that all our constraints are fulfilled. Let $E_m^- = E_m \backslash \{\emptyset\}$, that is we are excluding the all-zero settings as in equation (5.68), and consider the side constraints.

$$0 \leq x_n \leq 1 \quad ; \quad 0 \leq w_{m,S} \leq 1 \quad ; \quad \mathbb{1}^T \boldsymbol{w}_m = 1 \quad ; \quad \forall S \in E_m \tag{5.70}$$

$$0 \leq \frac{1}{\beta} h_n \leq 1 \quad ; \quad 0 \leq \frac{1}{\beta} u_{m,S} \leq 1 \quad ; \qquad \forall S \in E_m^- \tag{5.71}$$

If we consider the last equation, $\mathbb{1}^T \boldsymbol{w}_m$, and write it out.

$$w_{m,\{\emptyset\}} + \sum_{S \in E_m^-} w_{m,S} = w_{m,\{\emptyset\}} + \frac{1}{\beta} \sum_{S \in E_m^-} u_{m,S} = 1 \tag{5.72}$$

If we now choose $\beta = \max_m \sum_{S \in E_m^-} u_{m,S}$, then $0 \leq \frac{1}{\beta} \sum_{S \in E_m^-} u_{m,S} \leq 1$. This means that $w_{m,\{\emptyset\}}$ can be selected within the bound $0 \leq w_{m,\{\emptyset\}} \leq 1$ such that the sum indeed will be 1. The bounding constraints in equation (5.71) will also be fulfilled by this setting of β.

This means that a pseudocodeword is a scaled and normalized point in the polytope P. This is useful for proving the performance of decoding in the BEC.

5.6.2 Decoding in the BEC

The objective in this section is to show that the LP decoder fails, if and only if the received message is a stopping set. First we introduce three important theorems which will help us to show the performance in the BEC.

"The probability that the LP decoder fails is independent of the codeword that was transmitted" [5].

Because of this theorem, we can always assume that the *all-zero codeword* is transmitted when analyzing the behavior of the LP decoder. It is straightforward to show that the zero codeword has zero cost, which make analysis easier.

$$\boldsymbol{\lambda}^T \boldsymbol{x} = \boldsymbol{\lambda}^T \boldsymbol{0} = 0 \tag{5.73}$$

If we now consider multiple optima where two or more codewords $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ have same the cost.

$$\boldsymbol{\lambda}^T \boldsymbol{x}_1 = \boldsymbol{\lambda}^T \boldsymbol{x}_2 \tag{5.74}$$

If there are multiple optima, it is chosen that this corresponds to failure. We can then give the following theorem.

Given that the all-zero codeword was transmitted, the LP decoder will fail if and only if there is some point in P other than $x = 0^N$ with $\boldsymbol{\lambda}^T \boldsymbol{x} \leq 0$ [5].

This can also be rewritten in terms of a pseudocodeword.

Given that the all-zero codeword was transmitted, the LP decoder will fail if and only if there is some pseudocodeword $\boldsymbol{h} \neq 0^N$, with $\boldsymbol{\lambda}^T \boldsymbol{h} \leq 0$ [4, p.64].

Another important consequence of the all-zero codeword assumption when considering the BEC is shown in the Figure 5.7. All $\lambda_n \geq 0$

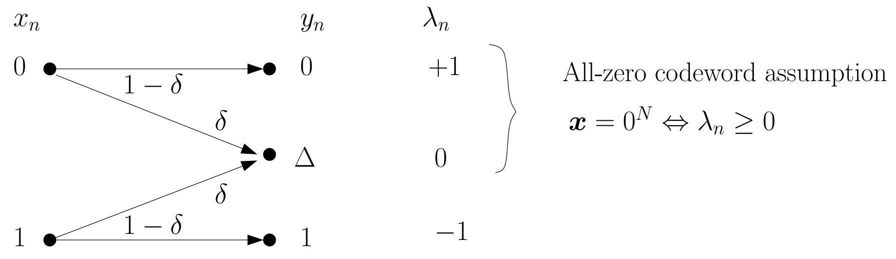

x_n
y_n
λ_n
0
1
$1-\delta$
δ
δ
$1-\delta$
0
Δ
1
$+1$
0
-1
All-zero codeword assumption
$\boldsymbol{x} = 0^N \Leftrightarrow \lambda_n \geq 0$

Figure 5.7: *By the all zero codeword assumption in the BEC, $\lambda_n \geq 0$*

We should now have the background to prove the following theorem.

In the BEC, there is a non-zero pseudocodeword with zero cost if and only if there is a stopping set [4, p.126-127].

The proofs follows the concept from [4, p.126-127], which are written out and changed a little for understanding. Start by considering the one way proof, if there is a zero cost pseudocodeword, there is a stopping set, i.e. $\sum_n \lambda_n h_n = 0$. If we define the set $\mathcal{S} = \{n, h_n > 0\}$, we can split the sum into two parts.

$$\sum_{n \in \mathcal{S}} \lambda_n h_n + \sum_{n \notin \mathcal{S}} \lambda_n h_n = 0 \tag{5.75}$$

We know that $h_n = 0 \ \forall \ n \notin \mathcal{S}$, because by the definition of pseudocodewords $h_n \geq 0$ and $h_n > 0 \ \forall \ n \in \mathcal{S}$. To bring it all to zero, $\lambda_n = 0 \ \forall \ \ n \in \mathcal{S}$. Then $y_n = \Delta \ \forall \ \ n \in \mathcal{S}$ follows directly from the Figure 5.7. This means that $\mathcal{E} \supseteq \mathcal{S}$, where $\mathcal{E} = \{n : y_n = \Delta\}$. Decoding fails if $\mathcal{E} \supseteq \mathcal{S}$. Let us now show that the definition of $\mathcal{S}$ is indeed a stopping set by our previous definition in section 4.5.2.

Suppose $\mathcal{S}$ is not a stopping set. This mean that there exist at least one check node m' which has only one edge connected to the $\mathcal{S}$, variable node n'. By the definition of $\mathcal{S}$, $h_{n'} > 0$. Consider now equation (5.68), if one $h_n > 0$ then there has to be another $h_l > 0$ because of the system of the matrix, where all columns have at least weight 2. By definition, then $l \ \in \mathcal{S}$, that is both l and n' are in the set, and connected to check equation m'. That is, m' is at least connected twice to $\mathcal{S}$, which contradict our initial statement that m' has only one connection. So, $\mathcal{S}$ is a stopping set.

Considering now the proof in the opposite direction, if there is a stopping set, there is a zero-cost pseudocodeword. We have that $\mathcal{E} \supseteq \mathcal{S}$. Set now $h_n > 0 \ \forall \ n \in \mathcal{S}$ and $h_n = 0 \ \forall \ n \notin \mathcal{S}$. Then

$$\sum_{n \in \mathcal{S}} \lambda_n h_n + \sum_{n \notin \mathcal{S}} \lambda_n h_n = 0 \tag{5.76}$$

because $\lambda_n = 0 \ \forall \ n \in \mathcal{S}$ and the setting $h_n = 0 \ \forall \ n \notin \mathcal{S}$. So if there is a stopping set, there is zero cost. Let us now show that these settings form a legal pseudocodeword, or in other word, we can find a set of $u_{m,S}$ that will allow the settings of h_n when using our previous definition of a pseudocodeword. We know that there is a stopping set, so for each check node there should be at least two or none variable nodes with $h_n > 0$. Consider now the example in (5.68) for one check equation. By proper selection of $u_{A,S}$ we can make $h_n > 0$ for at least two or none variable nodes. This is also possible for check constraints of other degrees. Since each $u_{m,S}$ is unique from check equation to check equation, it will always be possible to find a set of $u_{m,S}$ that will satisfy our initial settings of h_n. So, $\boldsymbol{h}$ is a pseudocodeword.

5.7 Multiple Optima in the BSC

This section is on the existence of multiple optima in the BSC. It will be derived how multiple optima occurs while decoding the $(7, 4, 3)$ Hamming code.

A leaf node is a variable node having degree one such that it is only connected with one check node. By example it was identified with the LP decoder, that if an error occurs in a leaf node of the $(7, 4, 3)$ Hamming code, the LP decoder may fail because of the occurrence of multiple optima. For an error anywhere else, decoding always succeeds. In this section we will investigate this observation.

We will start by convincing us that for ML decoding, a leaf node error is possible to decode. The $(7,4,3)$ Hamming code has $d_{min} = 3$, which means that we should be able to correct at minimum $t = \lfloor \frac{d_{min}-1}{2} \rfloor = 1$ error anywhere in the codeword [12]. So the loss of decoding performance is due to the degradation of the LP decoder compared to the ML decoder.

For the example, the alternative formulation is used for decoding and MOSEK 4 is used as a solver to yield two solutions, a basic point solution $\boldsymbol{x}_b^*$ and an interior point solution $\boldsymbol{x}_i^*$. If an error is placed in the leaf node for this code under the BSC, then we have,

$$\boldsymbol{\lambda}^T \boldsymbol{x}_b^* = \boldsymbol{\lambda}^T \boldsymbol{x}_i^* \qquad \text{where} \quad \boldsymbol{x}_b^* \neq \boldsymbol{x}_i^* \tag{5.77}$$

It states that both solutions have the same cost and there exist thereby multiple optima for this optimization problem. Both of them should be considered as correct solutions. However, there are more than two solutions to an error in the leaf node for the Hamming code. In fact there are infinitely many solutions as we will show now. Because there is equal cost in equation (5.77), then

$$\boldsymbol{\lambda}^T(\boldsymbol{x}_b^* - \boldsymbol{x}_i^*) = \boldsymbol{\lambda}^T \boldsymbol{r} = 0 \tag{5.78}$$

The vector of direction $\boldsymbol{r}$ is orthogonal to the cost vector λ. The problem of multiple optima could look like the Figure 5.8.

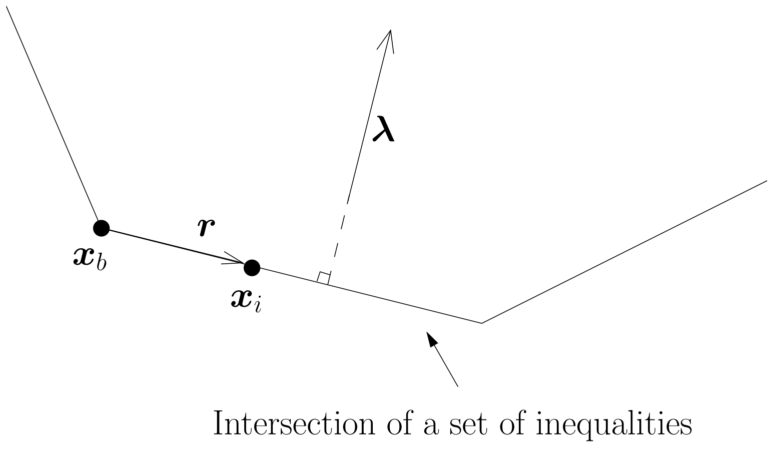


Figure 5.8: *Overview of multiple optima and MOSEK solutions.*

The reason why $\boldsymbol{x}_b^*$ is a vertex, is because by example $\boldsymbol{x}_b^*$ has always given an integer point solution. We now want to find the set of inequalities such that we can find the intersection that seems to be orthogonal to the cost vector $\boldsymbol{\lambda}$. Say that we are solving the following problem, where $\underline{\mathrm{G}}$ and $\boldsymbol{h}$ defines the inequalities from the alternative formulation.

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \underline{\mathbf{G}}\boldsymbol{x} \leq \boldsymbol{h} \\ & 0 \leq x_n \leq 1 \qquad \forall \quad n = 1,\ 2,\ \ldots N \end{aligned} \tag{5.79}$$

Let now each row equation of $\underline{\mathbf{G}}\boldsymbol{x} \leq \boldsymbol{h}$ be $\boldsymbol{g}_k^T\boldsymbol{x} \leq h_k$. The set of inequalities defining the intersection, is all the active constraints for the solution $\boldsymbol{x} = \boldsymbol{x}_i^*$. An active constraint is where equality holds such that the solution is on the bound of the polytope, i.e., the equations k for which $\boldsymbol{g}_k^T\boldsymbol{x}_i^* = h_k$ instead of just $\boldsymbol{g}_k^T\boldsymbol{x}_i^* \leq h_k$. All the active constraints are then the constraints for which one of the following equations holds.

$$\boldsymbol{g}_k^T\boldsymbol{x}_i^* = h_k \quad \text{or} \quad x_n = 0 \quad \text{or} \quad x_n = 1 \tag{5.80}$$

Let's now consider a specific example where $\boldsymbol{y} = \boldsymbol{x} \oplus \boldsymbol{e}$ (the BSC error model) $\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{e} = [0\ 0\ 0\ 0\ 0\ 0\ 1]^T$. Then $\boldsymbol{\lambda} = [1\ 1\ 1\ 1\ 1\ 1\ {-1}]^T$ and the solution given by LP decoding is $\boldsymbol{x}_i^* = [0\ 0\ 0\ 0.28\ 0.28\ 0.28\ 0.85]^T$. All the active constraints for this example are $k = 3, 4, 11, 12, 20$ and $n = 1, 2, 3$. The result is as in equation (5.81), where the active constraints are stacked in an augmented matrix..

$$\left.\begin{array}{l} \boldsymbol{g}_3^T\boldsymbol{x} = h_3 \\ \boldsymbol{g}_4^T\boldsymbol{x} = h_4 \\ \boldsymbol{g}_{11}^T\boldsymbol{x} = h_{11} \\ \boldsymbol{g}_{12}^T\boldsymbol{x} = h_{12} \\ \boldsymbol{g}_{20}^T\boldsymbol{x} = h_{20} \\ x_1 = 0 \\ x_2 = 0 \\ x_3 = 0 \end{array}\right\} = \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & h \end{array} \\ \left[\begin{array}{ccccccc|c} -1 & -1 & 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & -1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{5.81}$$

Equation (5.81) is then brought to row reduced echelon form.

$$\sim \left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{5.82}$$

There is one free variable, x_7. Selecting the free variable $x_7 = t$, the solution space is then a line.

$$\left.\begin{aligned} x_1 &= 0 \\ x_2 &= 0 \\ x_3 &= 0 \\ x_4 - \tfrac{1}{3}x_7 &= 0 \\ x_5 - \tfrac{1}{3}x_7 &= 0 \\ x_5 - \tfrac{1}{3}x_7 &= 0 \\ x_7 &= x_7 \end{aligned}\right\} \Leftrightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = t \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \\ 1 \end{bmatrix} = t\boldsymbol{s} \qquad t \in \mathbb{R} \tag{5.83}$$

The direction vector of this line is $\boldsymbol{s}$. This line is orthogonal to the cost vector since $\boldsymbol{\lambda}^T\boldsymbol{s} = 0 + 0 + 0 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} - 1 = 0$. So, all the points on this line have same cost, and thereby all are solutions to the LP decoding problem for this example. The same calculations can be done for the two other leaf node errors. The reason why the solution space is a line is because the intersection matrix is rank deficient by one (has rank one less than full rank). For 1 error anywhere else other than the leaf nodes, the intersection matrix has full rank, such that the solution is a single point, an unique solution.

It has not been possible to generalize the problem of multiple optima in the BSC for other codes. That is, identify under which conditions a linear code with a certain error pattern will have multiple optima when using LP decoding in the BSC.

5.8 Improving Performance using Redundant Constraints

In section 5.3.2, it was shown that LP decoding can sometimes fail by generating fractional solutions. The goal of this section is to improve the performance of the LP decoder. An algorithm using *redundant parity checks* will be explained that may change a fractional solution of LP decoder to a ML codeword after some iterations.

5.8.1 Background Information

Consider a transmitted vector $\boldsymbol{x}$ over a memoryless channel and its cost vector $\boldsymbol{\lambda}$. In section 5.5, the check constraint of the alternative formulation of the LP decoding problem was defined as,

$$\sum_{n\in V} x_n - \sum_{n\in\mathcal{N}(m)\setminus V} x_n \leq |V| - 1 \qquad \forall \quad V \subseteq \mathcal{N}(m), \quad |V| \text{ odd} \qquad 0 \leq x_n \leq 1 \tag{5.84}$$

With $\mathcal{N}(m)$ the neighbourhood of a check equation m and V is a subset of $\mathcal{N}(m)$.

The idea in this section is, *the LP decoder will begin with all constraints given by the parity checks and then afterwards, progressively, add more constraints by combining the parity checks to redundant parity checks until the ML codeword is found*. This method improves the error-rate performance of the LP decoder [13]. It might also reduces the complexity of the problem compared to the method which consists in adding from the beginning, all redundant parity checks to the constraints of the LP decoder.

Let us introduce an important term of this method, the *redundant parity check*. Consider the rows $\boldsymbol{h}_A$ and $\boldsymbol{h}_B$ of a parity check matrix. In equations (5.85) and (5.86), we have:

$$\boldsymbol{h}_A \otimes \boldsymbol{x} = 0 \quad \wedge \quad \boldsymbol{h}_B \otimes \boldsymbol{x} = 0 \tag{5.85}$$

$$\Updownarrow$$

$$\boldsymbol{h}_A \otimes \boldsymbol{x} = 0 \quad \wedge \quad \boldsymbol{h}_B \otimes \boldsymbol{x} = 0 \quad \wedge \quad (\boldsymbol{h}_A \oplus \boldsymbol{h}_B) \otimes \boldsymbol{x} = 0 \tag{5.86}$$

The new equation $(\boldsymbol{h}_A \oplus \boldsymbol{h}_B) \otimes \boldsymbol{x} = 0$ is redundant. So a redundant parity check is a check equation obtained by modulo 2 addition of some rows of the parity check matrix.

Moreover, the notion of *cut* is introduced. A cut can be considered as a half-plane which removes a fractional solution from the search domain of the polytope P. If the LP decoder gives a fractional solution, a cut can be used in order to remove this solution from the polytope. Thus, a new polytope is created and LP decoding can be executed again. We can proceed in this way until we find the ML codeword [13] or an upper bound of the number of iterations has reached.

If the solution $\boldsymbol{x}_j^*$ from iteration j fulfills the equation (5.87):

$$\sum_{n \in V} x_{j,n}^* - \sum_{n \in \mathcal{N}(m) \setminus V} x_{j,n}^* > |V| - 1 \tag{5.87}$$

then the equation (5.88)

$$\sum_{n \in V} x_n - \sum_{n \in \mathcal{N}(m) \setminus V} x_n \leq |V| - 1 \tag{5.88}$$

is a cut at $\boldsymbol{x}_j^*$. When a redundant parity check introduces a cut, it is called a *redundant parity check cut*. The following subsection will describe the algorithm which uses these redundant parity check cuts.

5.8.2 Algorithm of Redundant Parity Check Cuts

The algorithm has 4 steps:

1. If a fractional solution $\hat{\boldsymbol{x}}$ is output by the LP decoder for a given received cost vector $\boldsymbol{\lambda}$, then prune all the variable nodes in the factor graph corresponding to an integer code symbol. The resulting new factor graph can be called F.

2. Try to find a cycle in the factor graph F.

3. Add by modulo 2 the rows of the parity check matrix of the corresponding check nodes in the cycle.

4. If this new check equation introduces a cut, run the LP decoder and go to step 1. Else, go to step 2.

Step $1-4$ is guided by this theorem:

A redundant parity check constraint can introduce a cut only if the factor graph F contains cycle [13]. That is why in step 1, all the variables nodes having integer solutions are pruned from the factor graph in order to have a graph F with only variable and check nodes corresponding to the fractional solutions. Afterwards, in step 2, a cycle is identified. We know that if we find a cycle, we may be able to introduce a redundant parity check cut. Because we only may find a cut, we check for it in step 4 otherwise we search for another cycle.

5.8.3 Example

In this subsection the previous algorithm will be explained through an example (cf. the Matlab code *rpca_example.m* in the CD-ROM for this example).

Consider the $(7,4,3)$ Hamming code with the cost vector $\boldsymbol{\lambda} = [-\frac{7}{4}\ 1\ 1\ 1\ 1\ 1\ 1]^T$. This looks like that the codeword $\boldsymbol{x} = [0\ 0\ 0\ 0\ 0\ 0\ 0]^T$ was send across a BI-AWGNC with some error on the first symbol. The LP decoder outputs at first the fractional solution $\boldsymbol{x}_0^* = [1\ \frac{1}{3}\ 0\ \frac{1}{3}\ \frac{1}{3}\ 0\ 0]^T$. According to the first step of our algorithm, the variables nodes having integer solutions x_1, x_3, x_6 and x_7 have to be removed from the factor graph of this Hamming code. The Figure 5.9 corresponds to the new factor graph F_1:

From the second step, we decide to add check equation A and C because these participates in the selected cycle. The new parity check constraint is:

$$chk(A) \oplus chk(C) = x_1 \oplus x_2 \oplus x_6 \oplus x_7 \qquad (5.89)$$

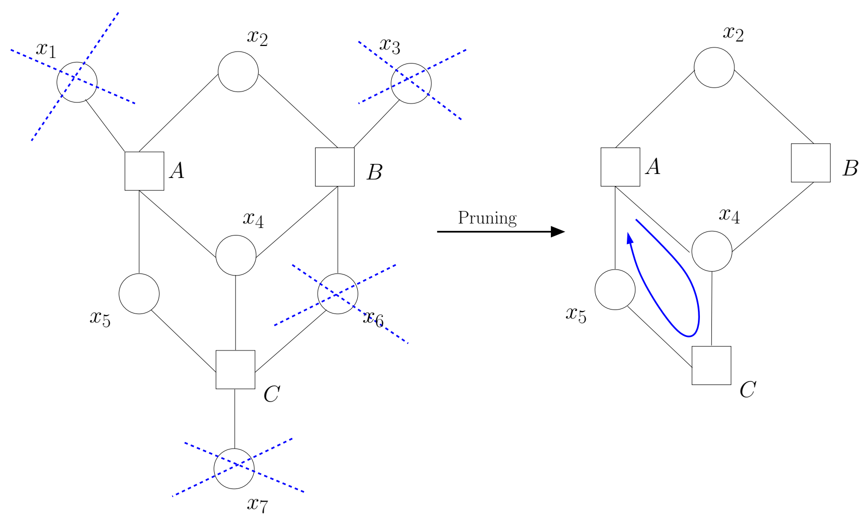


Figure 5.9: *The new factor graph F_1*

We verify next that this redundant check constraint generates a cut. This means that we need to verify that (5.84) is violated. In this case, we need to see, with $[x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7]^T = [1\ \frac{1}{3}\ 0\ \frac{1}{3}\ \frac{1}{3}\ 0\ 0]^T$, if any of the following row equations are violated:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 & 1 & -1 \\ -1 & -1 & 0 & 0 & 0 & -1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 1 & 1 \\ -1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \tag{5.90}$$

The first row does not satisfy equation (5.84) because: $1(1) + \frac{1}{3}(-1) = \frac{2}{3} \nleq 0$. So, the redundant check constraint introduces a cut (cut 1 in the Figure 5.10).

We now add this redundant parity check cut to the constraints to the LP decoding problem. LP decoder is executed again, and again it gives a fractional solution. But this time a different one: $\boldsymbol{x}_1^* = [1\ \frac{2}{3}\ 0\ \frac{1}{3}\ 0\ \frac{1}{3}\ 0]^T$. Indeed we have introduced a cut, because the optimal solution is now another one. The solution is still fractional and therefore we again search for a new cut to remove the fractional solution $\boldsymbol{x}_1^*$.

By following the same steps 4 times, as it is shown in the Figure 5.10 and Table 5.1, the LP decoder finally outputs the ML codeword after the fourth cut. $\boldsymbol{x}_4^* = [0\ 0\ 0\ 0\ 0\ 0\ 0]^T$. The LP decoder succeeds with the improved algorithm of adding redundant parity check cuts to the original constraints.

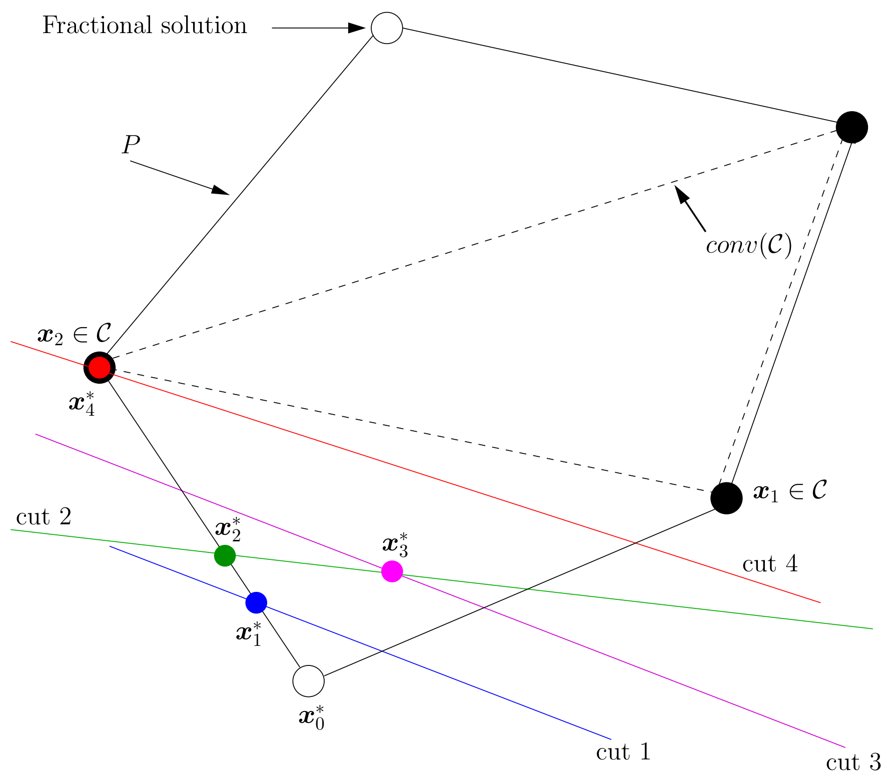


Figure 5.10: *Figure illustrates intuitively the algorithm of redundant parity check cuts for the example used.*

Iteration (i)	Checks in cycle	Redundant parity check cut	LP solution $\boldsymbol{x}_i^*$
0	——	——	$[1\ \frac{1}{3}\ 0\ \frac{1}{3}\ \frac{1}{3}\ 0\ 0]^T$
1	$chk(A) \oplus chk(C)$	$x_1 - x_2 - x_6 - x_7 \leq 0$	$[1\ \frac{2}{3}\ 0\ \frac{1}{3}\ 0\ \frac{1}{3}\ 0]^T$
2	$chk(B) \oplus chk(C)$	$x_2 - x_3 - x_5 - x_7 \leq 0$	$[1\ \frac{1}{2}\ 0\ \frac{1}{2}\ 0\ 0\ \frac{1}{2}]^T$
3	$chk(A) \oplus chk(B)$	$x_1 - x_3 - x_5 - x_6 \leq 0$	$[1\ \frac{1}{2}\ 0\ 0\ \frac{1}{2}\ \frac{1}{2}\ 0]^T$
4	$chk(A) \oplus chk(B) \oplus chk(C)$	$x_1 - x_3 - x_4 - x_7 \leq 0$	$[0\ 0\ 0\ 0\ 0\ 0\ 0]^T$

Table 5.1: *Table of different iterations for the algorithm of redundant parity check cuts.*

In the article [13], it can be seen that the algorithm of using redundant parity checks outperformed the normal LP decoding. This algorithm really improves the normal LP decoder and gives a satisfaction in the linear programming domain. Nevertheless, the performance of the improved algorithm with redundant parity check cuts goes only closer to the performance of the ML decoder. It never reaches it [4, p. 75-76].

Comparison 6

In this chapter, the similarities and differences between the message passing algorithm and the LP decoding are accentuated but not in detail. However, all the points that are discussed here have already been explained in detail in the previous chapters. So, the intention is to comprehend those points such that they can show the similarities and differences between the message passing algorithm and the LP decoding.

6.1 Optimal for Cycle Free Graphs

It was proved in section 4.3.1 and 4.4 that the sum-product and max-product/min-sum algorithm respectively are maximum likelihood for the cycle free factor graphs of the codes. However, sum-product algorithm is ML symbol-wise and max-product algorithm is ML block-wise. Then, it can also be reiterated that the sum-product algorithm may estimate a codeword which doesn't belong to the respective code $\mathcal{C}$ i.e., $\hat{\boldsymbol{x}} \notin \mathcal{C}$ whereas max-product will estimate a codeword from the respective code $\mathcal{C}$ i.e., $\hat{\boldsymbol{x}} \in \mathcal{C}$. However, the performance of the max-product/min-sum algorithm is bit worse (higher word-error rate) than the sum-product algorithm for the same signal to noise ratio (SNR).

LP decoding is also optimal for cycle free factor graph of a code. However, in this report, it was not proved to be optimal for a cycle free factor graph but it was taken for granted from the Jon Feldman's thesis [4][5]. LP decoder decodes block-wise like max-product/min-sum algorithm. LP decoder has got the ML certificate. Thus, if LP decoder outputs an integer solution, then it will always be a codeword from the respective code i.e., $\hat{\boldsymbol{x}} \in \mathcal{C}$.

6.2 Estimation/Scaling of Noise

It is quite interesting to know that the noise can't be scaled by any constant β in the sum-product algorithm whereas the noise can be scaled in the min-sum algorithm and LP decoder. The knowledge of the noise is necessary in the sum-product algorithm's check node operation because the check node operation uses box-plus summation of L-values. The scaling of noise means the scaling of log likelihood ratios (LLR/L-values). However, it is possible to scale the LLR in min-sum algorithm and LP decoder because the node operations in the min-sum algorithm at both variable and check nodes allow the scaling of LLR and the objective function of the LP decoder allows the scaling of LLR (λ).

6.3 Decoding in the BEC

It was proved in the section 4.5.2 that in the BEC, message passing algorithm fails if and only if a stopping set $\mathcal{S}$ exists among the erasure set $\mathcal{E}$ i.e., $\mathcal{E} \supseteq \mathcal{S}$. On the other hand, the LP decoding fails under BEC when there are multiple optima i.e., non-zero pseudocodeword with zero cost. So, in the BEC for LP decoder, there is a non-zero pseudocodeword with zero cost if and only if there is a stopping set which was also proved in the section 5.6. Thus the performance of both message passing algorithm and LP decoder are equivalent under the BEC.

6.4 Word Error Rate (WER) Comparison Under BSC

We have not done any simulations to measure WER of all three decoders. So, from the simulations done in [5], the performance of these three decoders under BSC can be construed and compared. The LP decoder performs better than the min-sum decoder but not as good as sum-product decoder. However, if the 3 decoders are compared to ML decoder, then they seem to have relatively similar performance for random codes. Moreover, in this article they have shown that the LP decoder surpasses the sum-product at very low noise levels.

6.5 Improvement by Adding Redundant Parity Checks

It was also proved in the section 5.8 that if the LP decoder renders a fractional solution then the fractional solution can be changed to an integer solution by adding redundant parity check constraints in the alternative formulation of LP decoding. The addition of redundant parity check constraints in LP will never degrade the performance whereas in message passing algorithm, it may degrade the performance [5].

6.6 Complexity

It's been known from the article [15] that the MPA decoder has overall linear complexity. However, the complexity of LP decoder is found to be approximately polynomial in time by performing a small simulation in MATLAB (cf. the MATLAB code *test_lp_complexity.m* in the CD-ROM). A LDPC code is randomly designed [14] with almost regular check node degree $d_c = 6$ and almost regular variable node degree $d_v = 3$ and $R = 1/2$. The time is considered from the call of the MOSEK 4 solver to the solver returns a solution using the tic-toc command in MATLAB. Averaged over 100 trials with a SNR = $E_s/N_0|_{\text{dB}} = 2, 6, 10$ dB. The specifications of the machine used for the calculation of the time are: 2 Ghz Pentium 4 CPU with 710 MB ram, running on a Linux kernel 2.6.3-7mdk, distribution Mandrake 10.0. The results, which are given in the Table 6.1, show that the computation time seems polynomial with the code length N and/or number of check equations M. The computational time increases as SNR decreases.

N	M	l_i/r_i	SNR = 2	SNR = 6	SNR = 10
250	125	$l_2 = 5/250, l_3 = 245/250$ $r_5 = 5/125, r_6 = 120/125$	0.35s	0.33s	0.30s
500	250	$l_2 = 12/500, l_3 = 488/500$ $r_5 = 12/250, r_6 = 238/250$	0.76s	0.70s	0.67s
1000	500	$l_2 = 6/1000, l_3 = 994/1000$ $r_5 = 6/500, r_6 = 494/500$	2.54s	2.27s	2.06s
2000	1000	$l_2 = 12/2000, l_3 = 1988/2000$ $r_5 = 12/1000, r_6 = 988/1000$	11.81s	10.71s	6.69s
4000	2000	$l_2 = 6/4000, l_3 = 3994/4000$ $r_5 = 6/2000, r_6 = 1994/2000$	78.16s	72.24s	67.49s

Table 6.1: *The computational time measured in seconds [s] for different SNR and design criteria for a LDPC code. l_i is the proportion of variable nodes of degree i. r_i is the proportion of check nodes of degree i (cf. appendix A).*

These results are found by solving the linear programming formulation which has the form,

$$\begin{aligned} \text{min.} \quad & \boldsymbol{\lambda}^T \boldsymbol{x} \\ \text{s.t.} \quad & \underline{\mathbf{G}}\boldsymbol{x} \leq \boldsymbol{h} \\ & 0 \leq x_n \leq 1 \qquad \forall \quad n = 1,\ 2,\ \ldots N \end{aligned} \tag{6.1}$$

where the matrix $\underline{\mathbf{G}}$ has dimension $M \cdot 2^{d_c - 1} \times N$ for a regular code. The dimension grows both in M and N. Further, the dimension is independent of d_v. The variable node degree only determines the block dependencies in the matrix $\underline{\mathbf{G}}$. For example, if $d_v = 3$, then each variable node is participating in 3 blocks of constraints. Each block could look like equation 5.61.

Conclusion 7

The goal of the project was basically to study two iterative methods to decode linear block and LDPC codes. These two iterative techniques were message passing algorithm and LP decoding. They were compared in the chapter 6.

LDPC codes were introduced in section 3.3. LDPC codes are sparse in nature and have the potential to approach Shannon limit. The decoding of LDPC codes can be done by the two algorithms which are studied in this project. A code can be represented by a factor graph such that the global code constraint can be factored into local code constraints defined by the parity check matrix of the code.

In the message passing algorithm, sum-product and min-sum algorithm were studied and expounded. The message passing algorithm operates in the factor graph and computes the marginal functions associated with the global code constraint. The message passing algorithm exchange the extrinsic messages along the edges of the factor graph. At the nodes, local decoding operations update the extrinsic messages according to the message update rules. The messages can be either in terms of probabilities or L-values. The sum-product algorithm is symbol-wise decoding whereas min-sum algorithm is block-wise decoding. However, the performance of the sum-product algorithm is better (lower word error rate) than the min-sum algorithm for fixed SNR. The min-sum algorithm can be considered as an approximation of the sum-product algorithm. If the factor graph is cycle free, then the message passing algorithm will be optimal (maximum likelihood) otherwise sub-optimal.

The LP decoder is defined by a cost function and the set of linear constraints. The optimization problem having addition and multiplication in $\mathbb{F}_2$ was reformulated such that all the operations are in $\mathbb{R}$. The LP constraints are formulated for each check equations of the parity check matrix of a code. The solution of the LP is always the vertex of the polytope. A solution of LP can be fractional too because of the relaxation of the LP. If the solution is integer, then it will be the ML codeword. So, the LP decoder has got the ML certificate property. The LP decoder is also ML block-wise for a cycle free factor graph. The LP decoder will succeed if and only if the lowest cost pseudocodeword is the transmitted codeword. The alternative formulation of LP without the indicator variable w was also studied and explained. Some interesting points were shown that for check node degree three, no variable bounding is needed. LP decoding of a $(7, 4, 3)$ Hamming code has shown that multiple optima exist under the BSC. Another aspect of the LP decoder is that it has the potential to improve the performance by the addition of the redundant parity check constraints.

The message passing algorithm under the BEC will fail if and only if a stopping set $\mathcal{S}$ exists among the erasure set $\mathcal{E}$ i.e., $\mathcal{E} \supseteq \mathcal{S}$. The LP decoder will fail in the BEC, if and only if there exists a stopping set. The performance of message passing algorithm and LP decoding is then equivalent under the BEC. Moreover, the overall decoding complexity of the message passing algorithm is linear whereas LP decoding seems polynomial by experiment.

Bibliography

[1] A Anastasopoulos, *A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution*, Global Telecommunications Conference, GLOBECOM'01 (2001).

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, *Near shannon limit error-correcting coding and decoding: Turbo codes*, Proc. IEEE Int. Conf. Commun. (ICC) (1993), 1064–1070.

[3] George B. Dantzig and Mukund N. Thapa, *Linear programming, 2: Theory and extensions*, Springer, 1997, ISBN: 0-387-98613-8.

[4] Jon Feldman, *Decoding error-correcting codes via linear programming*, Ph.D. thesis, Massachusetts Institute of Technology, 2003.

[5] Jon Feldman, Martin J. Wainwright, and David R. Karger, *Using linear programming to decode binary linear codes*, IEEE Transactions on Information Theory, 51(3), pp. 954–972. (2005).

[6] R. Gallager, *Low-density parity-check codes*, IEEE Trans. Inform. Theory **8** (1962), no. 1, 21–28.

[7] R.W. Hamming, *Error detecting and error correcting codes*, Bell Systems Technical **29** (1950), 147–160.

[8] Yunghsiang S. Han, *Introduction to binary linear block codes*, Graduate Institut of Communication Engineering, National Taipei University, 8–17.

[9] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Trans. Inform. Theory **47** (2001), 498–519.

[10] Ingmar Land, *Sipcom9-1: Lecture notes - advanced channel coding*, 2006.

[11] H.-A. Loeliger, *An introduction to factor graphs*, IEEE Signal Proc. Mag (2004), 28–41.

[12] Robert H. Morelos-Zaragoza, *The art of error correcting coding*, John Wiley & Sons,Ltd, 2002, ISBN: 0471 49581 6.

[13] Mohammad H. Taghavi N. and Paul H. Siegel, *Adaptive linear programming decoding*, (2006).

[14] Kim Nørmark and Ole Lodahl Mikkelsen, *Reducing the effect of cycles in decoding of ldpc codes*, Master's thesis, Aalborg University, 2006.

[15] T. Richardson and R. Urbanke, *The renaissance of gallager's low-density parity-check codes*, IEEE Commun. Mag. **41** (2003), no. 8, 126–131.

[16] Alexander Schrijver, *Theory of linear and integer programming*, John Wiley & Sons Ltd., 1986, ISBN: 0 471 90854 1.

[17] C.E. Shannon, *A mathematical theory of communication*, Bell Systems Technical **27** (1948), 379–423 and 623–656.

[18] Jeremy Thorpe, *Low-density parity-check codes*, 2006, Internet: `http://ldpc-codes.com`.

[19] Wikipedia.org , *Np-hard*, 2006, Internet: `http://en.wikipedia.org/wiki/NP-hard`.

[20] Wikipedia.org, *Convex hull*, 2006, Internet: `http://en.wikipedia.org/wiki/Convex_hull`.

[21] Wikipedia.org, *Low density parity-check code*, 2006, Internet: `http://en.wikipedia.org/wiki/LDPC`.

[22] Wikipedia.org, *Linear programming*, 2006, Internet: `http://en.wikipedia.org/wiki/linear_programming`.

Irregular Linear Codes

A

An irregular LDPC code is a code with different numbers of ones in each row and columns. Thus, new variables are defined for these irregular LDPC codes.

- l_i Proportion of variable nodes (left) of degree i
- r_i Proportion of check nodes (right) of degree i
- λ_j Proportion of edges going from variables nodes of degree j
- ρ_j Proportion of edges going from check nodes of degree j

The Figure A.1 can help to understand these relations:

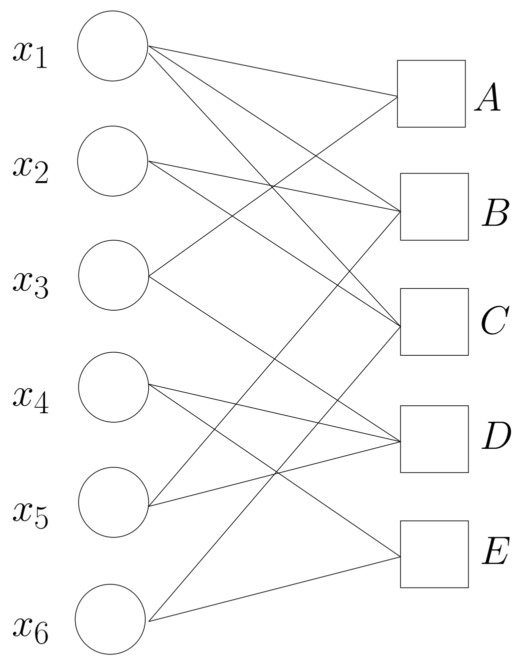


Figure A.1: *Irregular LDPC code.*

$$l_2 = 5/6\ ,\ l_3 = 1/6$$
$$r_2 = 2/5\ ,\ r_3 = 3/5$$
$$\lambda_2 = 10/13\ ,\ \lambda_3 = 3/13$$
$$\rho_2 = 4/13\ ,\ \rho_3 = 9/13$$

Two relations can be derived from the set of definitions:

- ∘ From the variable node point of view:

$$\lambda_i = \frac{i \cdot l_i}{\sum_j j \cdot l_j} \tag{A.1}$$

- ∘ For check nodes:

$$\rho_i = \frac{i \cdot r_i}{\sum_j j \cdot r_j} \tag{A.2}$$

Here is the proof for the first relation (A.1).

Let us define E as the total number of edges and N the number of variables. According to the definitions of λ_i and l_i, the number of variable nodes of degree i is equal to

$$l_i \cdot N = \frac{\lambda_i \cdot E}{i} \tag{A.3}$$

In fact, $\lambda_i \cdot E$ gives the total number of edges going from all variable nodes of degree i. If $\lambda_i \cdot E$ is divided by i (degree which corresponds to the number of edges going from each variable node), it will render the total number of variable nodes of degree i.

So,

$$\lambda_i = \frac{i \cdot l_i \cdot N}{E} \tag{A.4}$$

But, E could be defined as equal to:

$$E = \sum_k k \cdot l_k \cdot N = N \cdot \sum_k k \cdot l_k \tag{A.5}$$

The number of edges going from all variables nodes of degree k is $k \cdot l_k \cdot N$. When this number is summed over all the degrees, the total number of edges E is then obtained. By replacing E by its value, we obtain:

$$\lambda_i = \frac{i \cdot l_i}{\sum_k k \cdot l_k} \tag{A.6}$$

Similarly, the proof of the equation (A.2) is given below.

M is defined as the total number of check nodes.

$$E = \sum_k k \cdot r_k \cdot M = M \cdot \sum_k k \cdot r_k \tag{A.7}$$

According to the definitions of r_j and ρ_j, the number of check nodes of degree j is

$$r_j \cdot M = \frac{\rho_j \cdot E}{j} \tag{A.8}$$

So, by replacing E by its value, we obtain:

$$\rho_j = \frac{j \cdot r_j}{\sum_k k \cdot r_k} \tag{A.9}$$

The definition of an irregular LDPC code by the *the degree polynomials* can also be shown. So, the degree polynomials can be defined for the variable nodes and check nodes as,

$$\lambda(z) = \sum_i \lambda_i \cdot z^{i-1} \tag{A.10}$$

$$\rho(z) = \sum_i \rho_i \cdot z^{i-1} \tag{A.11}$$

The designed rate R_d for irregular LDPC code becomes:

$$R_d = 1 - \frac{\int_0^1 \rho(z) d_z}{\int_0^1 \lambda(z) d_z} \tag{A.12}$$

The equation (A.12) can be proved as given below.

Consider a graph with M (number of check nodes) N (number of variable nodes) and E (total number of edges). $\lambda_i \cdot E$ is the number of edges connected to the variable nodes of degree i. So, $\frac{\lambda_i \cdot E}{i}$ is the number of variable nodes of degree i. By summing over all the degrees, we obtain the total number of variable nodes:

$$\sum_i \frac{\lambda_i \cdot E}{i} = E \cdot \sum_i \frac{\lambda_i}{i} = N \tag{A.13}$$

Similarly, the number of check nodes

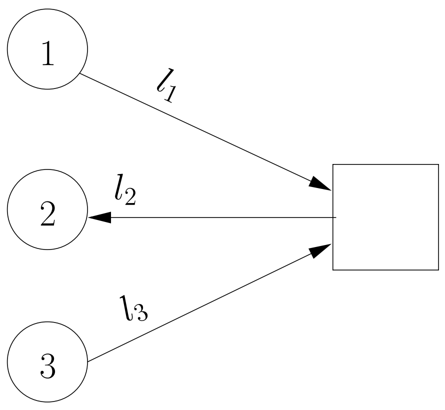
$$\sum_j \frac{\rho_j \cdot E}{j} = E \cdot \sum_j \frac{\rho_j}{j} = M \tag{A.14}$$

According to equation (3.16),

$$R_d = 1 - \frac{M}{N} = 1 - \frac{\sum_j \frac{\rho_j}{j}}{\sum_i \frac{\lambda_i}{i}} = 1 - \frac{\int_0^1 \rho(z) d_z}{\int_0^1 \lambda(z) d_z} \tag{A.15}$$

Proof of the Box-plus Operator B

The proof of the box-plus operator is derived here for the single parity check code having length 3 such that the outgoing extrinsic message l_2 of a check node A has box-plus summation of both incoming extrinsic messages l_1 and l_3 from the variable node x_1 and x_3 respectively, see figure B.1.



L−values

Figure B.1: *A factor graph of a single parity check code of length 3. The outgoing extrinsic message is l_2 and the two incoming messages are l_1 and l_3.*

It should be remembered that the probabilities and L-values have relation which are given in equations (4.43) and (4.44) as,

$$p^0 = \frac{e^{l/2}}{\left(e^{-l/2} + e^{l/2}\right)} \tag{B.1}$$

$$p^1 = \frac{e^{-l/2}}{\left(e^{-l/2} + e^{l/2}\right)} \tag{B.2}$$

where, $p^0 + p^1 = 1$.

So, from equation (4.93), the proof of the box-plus can be continued as,

$$l_2 = \ln\left(\frac{p^0_{ch,1}\cdot p^0_{ch,3} \;+\; p^1_{ch,1}\cdot p^1_{ch,3}}{p^0_{ch,1}\cdot p^1_{ch,3} \;+\; p^1_{ch,1}\cdot p^0_{ch,3}}\right)$$

The notations in the subscripts are dropped for the simplicity of the proof such that p defines $p_{ch,1}$ and q defines $p_{ch,3}$. Thus,

$$l_2 = \ln\left(\frac{p^0\cdot q^0 \;+\; p^1\cdot q^1}{p^0\cdot q^1 \;+\; p^1\cdot q^0}\right) \tag{B.3}$$

$$= \ln\left(\frac{\frac{e^{l_1/2}}{\left(e^{-l_1/2}+e^{l_1/2}\right)}\cdot\frac{e^{l_3/2}}{\left(e^{-l_3/2}+e^{l_3/2}\right)} \;+\; \frac{e^{-l_1/2}}{\left(e^{-l_1/2}+e^{l_1/2}\right)}\cdot\frac{e^{-l_3/2}}{\left(e^{-l_3/2}+e^{l_3/2}\right)}}{\frac{e^{l_1/2}}{\left(e^{-l_1/2}+e^{l_1/2}\right)}\cdot\frac{e^{-l_3/2}}{\left(e^{-l_3/2}+e^{l_3/2}\right)} \;+\; \frac{e^{-l_1/2}}{\left(e^{-l_1/2}+e^{l_1/2}\right)}\cdot\frac{e^{l_3/2}}{\left(e^{-l_3/2}+e^{l_3/2}\right)}}\right) \tag{B.4}$$

$$= \ln\left(\frac{e^{l_1/2}\cdot e^{l_3/2} \;+\; e^{-l_1/2}\cdot e^{-l_3/2}}{e^{l_1/2}\cdot e^{-l_3/2} \;+\; e^{-l_1/2}\cdot e^{l_3/2}}\right) \tag{B.5}$$

$$= \ln\left(\left(\frac{2}{2}\right)\cdot\left(\frac{e^{l_1/2}\cdot e^{l_3/2} \;+\; e^{-l_1/2}\cdot e^{-l_3/2}}{e^{l_1/2}\cdot e^{-l_3/2} \;+\; e^{-l_1/2}\cdot e^{l_3/2}}\right)\right) \tag{B.6}$$

$$= \ln\left(\frac{\left(\left(e^{l_1/2}+e^{-l_1/2}\right)\cdot\left(e^{l_3/2}+e^{-l_3/2}\right)\right) \;+\; \left(\left(e^{l_1/2}-e^{-l_1/2}\right)\cdot\left(e^{l_3/2}-e^{-l_3/2}\right)\right)}{\left(\left(e^{l_1/2}+e^{-l_1/2}\right)\cdot\left(e^{l_3/2}+e^{-l_3/2}\right)\right) \;-\; \left(\left(e^{l_1/2}-e^{-l_1/2}\right)\cdot\left(e^{l_3/2}-e^{-l_3/2}\right)\right)}\right) \tag{B.7}$$

$$= \ln\left(\frac{1 \;+\; \frac{\left(\left(e^{l_1/2}-e^{-l_1/2}\right)\cdot\left(e^{l_3/2}-e^{-l_3/2}\right)\right)}{\left(\left(e^{l_1/2}+e^{-l_1/2}\right)\cdot\left(e^{l_3/2}+e^{-l_3/2}\right)\right)}}{1 \;-\; \frac{\left(\left(e^{l_1/2}-e^{-l_1/2}\right)\cdot\left(e^{l_3/2}-e^{-l_3/2}\right)\right)}{\left(\left(e^{l_1/2}+e^{-l_1/2}\right)\cdot\left(e^{l_3/2}+e^{-l_3/2}\right)\right)}}\right) \tag{B.8}$$

(B.9)

It should be noted that

$$\tanh(Z) = \frac{e^Z - e^Z}{e^Z + e^Z} \tag{B.10}$$

So, the formulation of $\tanh(l/2)$ can be seen in the above equation (B.8).

Thus, this equation simplifies as,

$$l_2 = \ln \frac{\Big(1 + \tanh(l_1/2) \cdot \tanh(l_3/2)\Big)}{\Big(1 - \tanh(l_1/2) \cdot \tanh(l_3/2)\Big)} \tag{B.11}$$

As it is known that

$$\tanh^{\text{-1}}(Z) = \frac{1}{2} \cdot \ln \left(\frac{(1+Z)}{(1-Z)} \right) \tag{B.12}$$

Hence, the relation of $\tanh^{\text{-1}}$ is very conspicuous in the equation (B.11) such that

$$l_2 = 2 \tanh^{\text{-1}} \left(\tanh \left(\frac{l_1}{2} \right) \cdot \tanh \left(\frac{l_3}{2} \right) \right) \tag{B.13}$$

This relation of l_2 with l_1 and l_3 is also represented as [10],

$$l_2 = l_1 \boxplus l_3 \tag{B.14}$$

Now, If the equation (B.6) is re-considered, then

$$l_2 = \ln \left(\left(\frac{2}{2} \right) \cdot \left(\frac{e^{l_1/2} \cdot e^{l_3/2} \ + \ e^{-l_1/2} \cdot e^{-l_3/2}}{e^{l_1/2} \cdot e^{-l_3/2} \ + \ e^{-l_1/2} \cdot e^{l_3/2}} \right) \right) \tag{B.15}$$

$$= \ln \left(\left(\frac{2}{2} \right) \cdot \left(\frac{e^{(l_1+l_3)/2} \ + \ e^{-(l_1+l_3)/2}}{e^{(l_1-l_3)/2} \ + \ e^{-(l_1-l_3)/2}} \right) \right) \tag{B.16}$$

As it is known that

$$\cosh(Z) = \frac{e^Z + e^{-Z}}{2} \tag{B.17}$$

Then, from equation (B.16), it can be easily seen that

$$l_2 = \ln \left(\frac{\cosh\left((l_1 + l_3)/2 \right)}{\cosh\left((l_1 - l_3)/2 \right)} \right) \tag{B.18}$$