SOURCE AND CHANNEL CODING USING FOUNTAIN CODES

THÈSE N^{O} 4488 (2009)

PRÉSENTÉE LE 18 DÉCEMBRE 2009

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE D'ALGORITHMIQUE

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Bertrand Ndzana NDZANA

acceptée sur proposition du jury:

Prof. R. Urbanke, président du jury
Prof. M. A. Shokrollahi, directeur de thèse
Dr M. Fresia, rapporteur
Dr G. Shamir, rapporteur
Prof. E. Telatar, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2009

Je dédie ce travail à tous mes proches et plus particulièrement A

Caroline Menyé, ma compagne et nos enfants

Laurent Ndzana Ahanda, mon feu père, imprescriptiblement mon héros
Anastasie Ongola Bougou, ma mère
Mes soeurs et frères

Antoinette Claude Edzimbi, ma grand-tante maternelle et sa progéniture

Acknowledgements

First of all, I am extremely grateful to my PhD[1] thesis advisor, *Prof. Amin Shokrollahi*, for his unwavering professional and personal support. On the professional level, Amin is the most brillant scientist I have interacted with, and I am lucky to have had his guidance and his help regarding both practical and theoretical problems that I encountered during my work. I would also like to acknowledge his efforts to improve my presentation and writing skills which were lacking before. On a personal level, I would like to express my gratitude to him, for his understanding and kindness during difficult times. It would not have been possible to write this thesis without his insights and contributions.

I am also very grateful to *Prof. Andrew Eckford* for supervising and providing me with the guidance and support that I needed during my internships at York university. I would like to thank *Dr. Gil Shamir* for his suggestions and his help that contributed greatly to this work. I also wish to thank *Prof. Giuseppe Caire* and *Dr. Maria Fresia* who have taken time to provide advices on my work and for the various interesting discussions that we had. I would also like to thank *Dr. Harm Cronie* with whom I had many fruitful discussions and collaboration during the last part of my thesis period.

I thank a great deal *Prof. Rüdiger Urbanke*, *Prof. Emre Telatar*, *Dr. Maria Fresia* and *Dr. Gil Shamir* for kindly agreeing to be in my thesis committee and acknowledge the work they have put into reading my thesis and providing constructive comments.

My stay in Lausanne during my PhD was comfortable mainly due to the care taken by *Mrs Natascha Fontana*. She has helped me during my doctoral studies with all administrative issues, from finding a flat to printing my thesis. I also wish to thank *Dr. Giovanni Cangiani* and *M. Damir Laurenzi* for solving problems I had with computers. I would like to thank them for their kindness and friendship.

[1]This work was supported by Grant 228021-ECCSciEng of the European Research Council and by Grant 200020-115983/1 of the Swiss National Fund.

On a more personal level, thanks to the students and my colleagues from the school of computer and communication sciences at EPFL. In particular, *Amin Shokrollahi, Emre Telatar, Rüdiger Urbanke, Serge Vaudenay, Philippe Oechslin, Bixio Rimoldi, Muriel Bardet, Francoise Behn, Yvonne Huskie, Chantal Francois, Jean-Pierre Wassmer, Jean-Cédric Chappelier, Natascha Fontana, Evelyn Duperex, Corinne Degott, Giovanni Cangiani, Damir Laurenzi, Cyril Measson, Dinkar Vasudevan, Abdelaziz Amraoui, Henri Pfister, Andrew Brown, Lorenz Minder, Christina Fragouli, Gérard Maze, Mehdi Molkaraie, Mahdi Cheraghchi, Frédéric Didier, Frédérique Oggier, Payam Pakzad, Pooya Pakzad, Masoud Alipour, Harm Cronie, Bertrand Meyer, Hesam Salavati, Etienne Perron, Christine Neuberg, Raj Kumar, Luoming Zhang, Ghid Maatouk, Eren Sasoglu, Ayfer Ozgur, Marius Kleiner, Vojislav Gajic, Soheil Mohajer, Satish Korada, Shrinivas Kudekar, Nicolas Macris, Sanket Dusad, Vishwambhar rathi, Peter Berlin, Jasper Goseling, Manfred Hauswirth, Amina Chebira, Olivier Roy, Olivier Levêque, Roman Schmidt, Iryna Andriyanova, Shahram Yousefi* and *Sofiane Sarni.*

I would like to thank some friends who were close during all these years of "exile": *Franck Felgbo, Henoc Agbota, Olivier Mouangué, Mehdi Touati, Alain Golay, Paulette Akamba, Ignace Wafo, Etoa Christian, Mbarga Thierry, Kana Stéphane, Rodrigue Deuboué, Fred Letang, Simon Olinga, Jean-Marie Abomo, Loicq Bakay, Stella Foaleng, Claire Burdet, Sofiane Sarni, Delphine Aebi, Nadège Chevallier, Sarah Richard, Christophe Ukegbu-oji Bassey, Anouk Zbinden, Katia Serdyuk, Charlotte Crettenand, Dorine Rouiller, Eric Mbomo, Clyde Fohouo, Lambert Sonna, Murielle Tiambo, Essomba Hervé and his wife Lucie, my aunt Tang Jeanne and her offspring, Krystel Tchoungui, Rufine Fankam, Marie Victorine Onana* and *Patience Eyenga.*

I will never thank enough my late father *Laurent Ndzana* and my mother *Anastasie Ongola* for providing me with a solid education and for their invaluable support. I would like to thank my sisters and my brothers for their constant encouragement during this long endeavor. I wish to express my gratitude to my grandaunt *Antoinette Edzimbi* and her progeny, for their precious support during my stay in Switzerland.

Finally, I feel greatly indebted to my fiancée *Caroline Menyé* for all her trust, love and encouragement especially at the end of my PhD. The final year of this work was enriched by your influence in my life.

Contents

Abstract

The invention of Fountain codes is a major advance in the field of error correcting codes. The goal of this work is to study and develop algorithms for source and channel coding using a family of Fountain codes known as Raptor codes.

From an asymptotic point of view, the best currently known sum-product decoding algorithm for non binary alphabets has a high complexity that limits its use in practice. For binary channels, sum-product decoding algorithms have been extensively studied and are known to perform well. In the first part of this work, we develop a decoding algorithm for binary codes on non-binary channels based on a combination of sum-product and maximum-likelihood decoding. We apply this algorithm to Raptor codes on both symmetric and non-symmetric channels. Our algorithm shows the best performance in terms of complexity and error rate per symbol for blocks of finite length for symmetric channels.

Then, we examine the performance of Raptor codes under sum-product decoding when the transmission is taking place on piecewise stationary memoryless channels and on channels with memory corrupted by noise. We develop algorithms for joint estimation and detection while simulataneously employing expectation maximization to estimate the noise, and sum-product algorithm to correct errors. We also develop a hard decision algorithm for Raptor codes on piecewise stationary memoryless channels. Finally, we generalize our joint LT estimation-decoding algorithms for Markov-modulated channels.

In the third part of this work, we develop compression algorithms using Raptor codes. More specifically we introduce a lossless text compression algorithm, obtaining in this way competitive results compared to the existing classical approaches. Moreover, we propose distributed source coding algorithms based on the paradigm proposed by Slepian and Wolf.

Keywords : Raptor codes, LT codes, non-binary, Sum-Product, Maximum-Likelihood, Piecewise Stationary Memoryless Channels, Gilbert-Elliot channels, Slepian-Wolf

Résumé

L'invention des Fountain codes est une véritable révolution dans dans le domaine des codes correcteurs d'erreurs. Le but de ce travail est d'étudier et de développer des algorithmes de codage de source et de canal à l'aide d'une famille de codes Fountain appelée Raptor.

Du point de vue asymptotique, le meilleur algorithme actuel de décodage sum-product pour des alphabets non binaires a une complexité considérable qui limite son utilisation pour des problèmes pratiques de communication. Dans la littérature, les algorithmes de décodage sum-product pour des alphabets binaires connaissent un succès considérable pour des canaux à alphabets binaires. Dans la première partie de ce travail, nous développons un algorithme de décodage basé sur la combinaison des algorithmes conjoints sum-product et maximum-likelihood utilisant des codes binaires dédiés à des canaux de communication non-binaires. Nous appliquons cet algorithme à des codes Raptor sur des canaux de communication symmétrique et a-symmétrique. Nous constatons que notre algorithme présente une meilleur performance en terme de complexité et de taux de correction d'erreurs par symbole pour des blocs de données de taille finie.

Nous examinons ensuite les performances des codes Raptor avec l'algorithme sum-product lorsque la transmission se déroule sur des canaux sans mémoire stationnaires par morceau et sur des canaux avec mémoire avec l'hypothèse que le bruit est inconnu. Nous avons ainsi developpé des algorithmes d'estimation et de détection conjoints utilisant simultanément la maximisation de la moyenne afin d'estimer le bruit et le décodage sum-product afin de corriger les erreurs. Nous développons aussi des algorithmes Raptor de décodage sum-product quantifié du point de vue asymptotique pour des canaux sans mémoire stationnaires par morceau. Finalement, nous développpons des algorithmes d'estimation et de détection pour des canaux à mémoire de type markovien appelés Gilbert-Elliot.

Dans la troisième partie de ce travail, nous développons des algorithmes de compression sans pertes utilisant des codes Raptor. Plus spécifiquement nous avons introduit un algorithme sans pertes de compression de textes, obtenant ainsi des résultats compétitifs et robustes en comparaison aux algorithmes classiques existants. Par ailleurs, un algorithme de codage de sources distribuées

est proposé suivant le paradigme établi par Slepian et Wolf.

Mots-clés: Raptor codes, LT codes, Non-Binaire, Sum-Product, Maximum-Likelihood, Canaux sans mémoire stationnaires par morceau, Canaux de Gilbert-Elliot, Slepian-Wolf

List of Figures

List of Tables

List of Acronyms

Acronym	Description
SP	Sum-Product
LT	Luby Transform
PSMC	Piecewise Stationary Memoryless Channel
GEC	Gilbert-Elliot Channel
BWT	Burrows-Wheeler Transform
CLID	Closed-Loop Iterative Doping
BEC(p)	Binary Erasure Channel with erasure probability p
BSC(p)	binary symmetric channel with crossover probability p
$q-$SC(p)	q-ary symmetric channel with parameter p
LLR	Log-Likelihood Ratio
BIMSC	Binary Input Memoryless Symmetric Channel
ML	Maximum Likelihood
PAM	Pulse Amplitude Modulation
EM	Expectation Maximization
BP	Block Partitioning
RD	Recursive Decision
PSMC	Piecewise Stationary Memoryless Channel
MM-BSC	Markov-Modulated Binary Symmetric Channel
GE	Gilbert-Elliott
BWT	Burrows-Wheeler Transform
BWCA	Burrows-Wheeler compression algorithm
GST	Global Structure Transformation
EC	Entropy Coding
WFC	Weighted Frequency Count
IFC	Incremental Frequency Count
FC	Fountain Coder
CLID	Closed-Loop Iterative Doping
BID	Blind Iterative Doping

List of Notations

Symbol	Description
$\mathfrak{I}_a^b$	set $\{a, a+1, \cdots, b\}$
m	strict positive integer
q	integer equal to 2^m
GF(q)	Galois field of order q
k	number of input symbols
N	number of output symbols
$f^{'}(x)$	derivative of $f(x)$ with respect to x
Ω_i	probability that a randomly chosen output node is of degree i
Ω	output node degree distribution
ω_i	probability that a randomly chosen edge is connected to an output node of degree i
ω	output edge degree distribution
I_j	probability that a randomly chosen input node is of degree j
I	input node degree distribution
ι_j	probability that a randomly chosen edge is connected to an input node of degree j
ι	input edge degree distribution
R	rate of the code
C or $[n, k]$	linear code of block length n and dimension k
α	average degree of an input symbol
β	average degree of an output symbol
ϵ	reception overhead
$\mathcal{C}_q$	q-ary channel
Cap($\mathcal{C}_q$)	capacity of a q-ary channel $\mathcal{C}_q$
Pr[A]	probability of event A occuring by itself
$\widehat{x}$	estimate of x
H^{-1}	inverse matrix of H
inb^i	input node of degree i
$outb^j$	output node of degree j

$M_{\mathcal{C}}$	LLR associated with channel $\mathcal{C}$
Φ	Set of linear forms from GF(2^m) to GF(2)
C_Φ	linear $[n, m]_2$-code defined by Φ
$G(C_\Phi)$	generator matrix of C_Φ
$\dim\langle A\rangle$	dimension of space generated by elements of A
$\dim(A)$	dimension of vector space A

1 Introduction

1.1 Outline

The invention of **Fountain codes** is a major event in the field of error correcting codes during the last decade. Raptor codes are a family of Fountain codes proven to have good asymptotic and non-asymptotic properties.
The goal of this work is to study and develop algorithms for source and channel coding using Raptor codes.
In what follows we give the outline of the thesis and provide a road map of our major results.

Chapter 2 We introduce a class of decoding algorithms for binary Raptor codes used for transmission over q-ary channels, where $q = 2^m$. The algorithms provide a trade-off between complexity and decoding capability. Whereas the running time of the q-ary sum-product algorithms is $m2^m$ times that of its binary counterpart, in our case the complexity factor can be chosen between m and 2^m, depending on the error-correction capability required. As such, the running time can be much better than the q-ary sum-product algorithm. The main idea behind our algorithm is to apply an appropriate set of GF(2)-linear forms of GF(q) to the Raptor code to obtain a set of binary codes which can be decoded independently in parallel. After a prescribed number of iterations, for each of the input symbols of the Raptor code and each of the linear forms, an estimate is obtained on the probability that this linear form applied to the input symbol is zero. By gathering these probabilities and performing a maximum-likelihood decoding on a suitable code of very small blocklength, we are able to obtain a good estimate of the value of the input symbol. Simulation results are provided for families of q-ary symmetric channels and non-symmetric channels that show the performance of our decoding

algorithms.

Chapter 3 Two fixed per-information symbol complexity lossless source coding algorithms are modified for estimation and incremental Luby Transform (LT) decoding over Piecewise Stationary Memoryless Channels (PSMC's) with a bounded number of abrupt changes in channel statistics. In particular, as a class of PSMC's, binary symmetric channels are considered with a crossover probability that changes a bounded number of times with no repetitions in the statistics. Simulation results illustrate the benefits of using our algorithms, both in terms of probability of error and in terms of redundancy.
We also investigate the performance of Raptor codes using Gallager's majority decoding algorithm on the PSMC. We obtain equations which relate the error probability to the ouput node degree distribution.
In the last part of this chapter, we adapt the sum-product algorithm to Raptor estimation-decoding over a class of Markov-modulated channels called Gilbert-Elliot channels.

Chapter 4 We present an algorithm in a purely lossless text compression setting based on Raptor codes and the Burrows-Wheeler Transform. The algorithm proceeds as follows. First we apply a Text Preprocessing; secondly, after applying the Burrows-Wheeler Transform (BWT), we reduce the number of symbols by applying a run length encoding scheme; then, we transform the local context of the symbols into a global context by an incremental frequency count stage; and finally, we separately encode the run-length data stream with an entropy coder and the incremental frequency count-index stream with a layered Fountain Coder. The proposed scheme follows the Closed-Loop Iterative Doping (CLID) algorithm together with the multilevel stage decoding Sum-Product at the fountain coder stage. Our algorithm offers encouraging compression rates performance for large files.
We also present a scheme to the multiple user Slepian-Wolf problem in a certain part of the achievable region using Raptor codes.

1.2 Notation and Definitions

Throughout the manuscript, we use the following notation.

We use $\mathcal{C}_q$ to denote a noisy channel for q-ary alphabets, BEC(p) to denote the binary ersaure channel with erasure probability p, BSC(p) to denote the binary symmetric channel with crossover probability p, $q-$SC(p) to denote the q-ary symmetric channel with parameter p, where $0 \leq p \leq 1$. If A is an event, Pr[A] is the probability of event A occurring by itself. Let x denote the row vector $(x_1, \cdots, x_n)$ of length n, $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_n)$ is a row vector estimate of x. For any set F, F^k denotes the set of row vectors $u = (u_1, \cdots, u_k)$ where for $j = 1, \cdots k$, we have $u_j \in F$. If H is an invertible $k \times k$ matrix, H^{-1} is the

inverse matrix of H. GF(q) is the Galois field of order q, typically, $q = 2^m$ and $m > 0$. If a and b denote two integers, where $a \leq b$, then we denote $\mathfrak{I}_a^b$ as the set of integers $\{a, a+1, \cdots, b\}$. If A is a set, $\dim\langle A\rangle$ is the dimension (i.e. the cardinality of a basis) of the space generated by elements of A. If A is a space itself, $dim(A)$ is the deimension of A. We denote by $\mathbf{1}()$ an indicator function defined by

$$\mathbf{1}(a = b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

A PAM (Pulse Amplitude Modulation) signal constellation is a constellation with a uniform spacing and a uniform distribution on the constellation symbols.

1.3 Fountain Codes

Fountain codes are codes for which the rate is not fixed in advance, and new code symbols can be generated as needed. This family of codes was originally [4] designed for transmission over computer networks. These codes are used to transmit information over channels for which the noise level is not known in advance. They can be used for transmission over BEC(p) when the parameter p is unknown. A Fountain code produces a potentially limitless stream of output symbols $z_1, z_2, \cdots$ for a given sequence of input symbols $x_1, \cdots, x_k$ of length k.

A decoding algorithm for a well designed Fountain code can recover the original k input symbols from any set of N output symbols with high probability, where N is close to k. In what follows we assume that symbols are binary.

1.3.1 LT-Codes and Raptor Codes

LT codes are one of the first classes of Fountain codes [34], in which each coded bit is the exclusive-or of a randomly selected subset of the input bits. The number of input bits in this set is called the degree of the output bit.

An LT code has parameters (k, Ω), where $k > 0$ is the length of input sequence and Ω denotes a probability distribution on the set $\mathfrak{I}_1^k$. The decoding graph of an LT code of length k with parameters (k, Ω) is a bipartite graph (as can be seen in Fig. 1.1) with k nodes on one side (called input nodes which correspond to the input bits) and N nodes on the other side (called output nodes which correspond to output bits), where $N > 0$. We sometimes say that a node has value v if the bit corresponding to that node has value v. There is an edge from each output node to all those input nodes whose sum is equal to the value of the output node before transmission.

Let $D_{in} \in \mathfrak{I}_1^N$ be the maximum input node degree, $D_{out} \in \mathfrak{I}_1^k$ the maximum output node degree, I_i be the probability that a randomly chosen input node

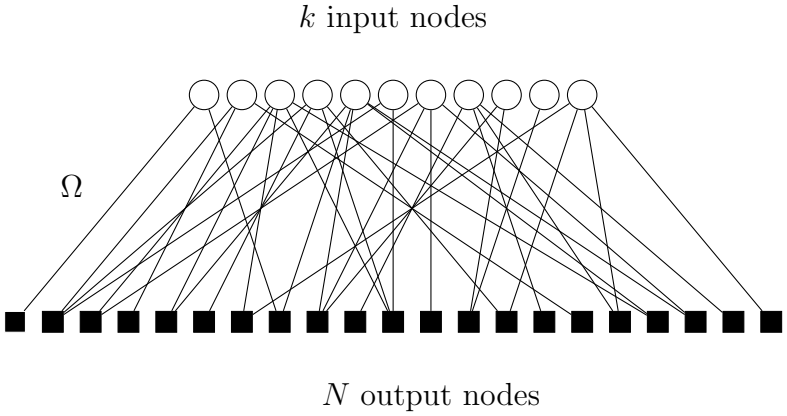


Figure 1.1: Decoding graph of an LT code

is of degree i and Ω_j be the probability that a randomly chosen output node is of degree j, where $i \in \mathcal{I}_1^{D_{in}}$ and $j \in \mathcal{I}_1^{D_{out}}$.

$I = (I_1, I_2, \cdots, I_{D_{in}})$ and $\Omega = (\Omega_1, \Omega_2, \cdots, \Omega_{D_{out}})$ are called degree distributions and can be represented by their generating functions :

$$I(x) = \sum_{i=1}^{D_{in}} I_i x^i, \Omega(x) = \sum_{j=1}^{D_{out}} \Omega_j x^j.$$

Let ι_i be the probability that a randomly chosen edge is connected to an input node of *degree* i and ω_j be the probability that a randomly chosen edge is connected to an output node of *degree* j in the decoding graph.

$\iota = (\iota_1, \iota_2, \cdots, \iota_{D_{in}})$ and $\omega = (\omega_1, \omega_2, \cdots, \omega_{D_{out}})$ are called degree distributions and can also be represented by their generating functions as

$$\iota(x) = \sum_{i=1}^{D_{in}} \iota_i x^{i-1}, \omega(x) = \sum_{j=1}^{D_{out}} \omega_j x^{j-1}.$$

Degree distributions $\Omega(x)$, $\omega(x)$, $I(x)$ and $\iota(x)$ are called output node degree distribution, output edge degree distribution, input node degree distribution and input edge degree distribution respectively; they are associated as:

$$\iota(x) = \frac{I'(x)}{I'(1)}, \omega(x) = \frac{\Omega'(x)}{\Omega'(1)},$$

where $I'(x)$ and $\Omega'(x)$ are the derivatives of $\Omega(x)$ and $I(x)$, with respect to x, respectively.

Let α be the average degree of an input symbol and β be the average degree of an output symbol. The nominal rate of the code is

$$R \triangleq \frac{k}{N} = \frac{\beta}{\alpha}.$$

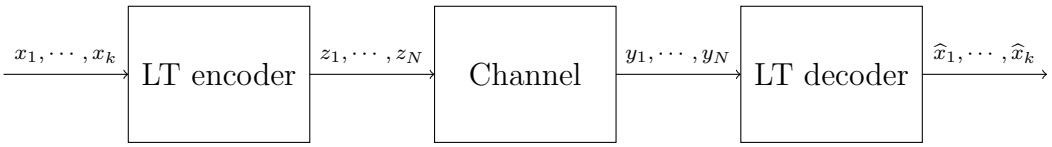


Figure 1.2: LT communication system for binary alphabets.

By assuming that the number of output nodes is large, input node degree distribution is binomial by construction and can be approximated by a Poisson distribution with parameter α (*Proposition 1* of [17]). Thus, the input node degree distribution is replaced by

$$I(x) = e^{\alpha(x-1)}$$

and the associated input edge degree distribution is

$$\iota(x) = \frac{I'(x)}{I'(1)} = e^{\alpha(x-1)}.$$

A row vector of k bits is encoded using a specific LT code $x = (x_1, \cdots, x_k)$ into the row vector $z = (z_1, \cdots, z_N)$.

To generate an output bit, the encoder works as follows.

Algorithm 1. *LT encoder rule*
Input: Row vector of input bits $x = (x_1, \cdots, x_k)$ *and the output node degree distribution* Ω.
Output: Row vector of output bits $z = (z_1, \cdots, z_N)$.
For $l = 1, \cdots N$, *do*

1. *Randomly choose the degree* j *of the output bit using* Ω.
2. *Choose uniformly at random* j *distinct input bits.*
3. *The value of the output bit* z_l *is the exclusive-or of these* j *input bits.*

As shown in Fig. 1.2, the vector z is then transmitted over a particular channel $\mathcal{C}_2$. We denote by $y = (y_1, \cdots, y_N)$ the received row vector and by $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ an estimate of x obtained after the decoding. The transmission is done bit-by-bit. The value of N does not need to be known in advance.

The performance of LT codes with respect to a given decoding algorithm is measured in terms of the error rate as a function of the reception overhead denoted by ϵ. The reception overhead is the number of output bits that the decoder needs to collect in excess of the absolute minimum in order to recover the input bits with high probability. The decoder collects output bits and estimates for each received output bit the amount of information in that bit. This measure of information can be obtained from the log-likelihood ratio (LLR) of the received bit. The receiver stops collecting output bits as soon as

the accumulated information carried by the observed channel outputs exceeds $(1+\epsilon)k$, where ϵ is the overhead associated with the LT code, and k is the number of input bits. For a given channel $\mathcal{C}_2$ with capacity $\mathrm{Cap}(\mathcal{C}_2)$, the rate R is associated to ϵ as

$$R = \frac{\mathrm{Cap}(\mathcal{C}_2)}{1+\epsilon}.$$

Raptor codes [47] are an extension of the family of LT codes. A Raptor code is a code formed by concatenating a given linear code C (usually a high-rate LDPC code) of block length $n > 0$ and dimension $k > 0$, with an LT code [47]. The Raptor code is encoded by first encoding k information bits into an n-bit codeword in C. Subsequently, each of the n bits is re-encoded as an LT codeword. The code C is called the pre-code of the Raptor code.

Since the code C may be trivial (i.e., the output sequence is always equal to the input sequence), an LT code is a special case of a Raptor code. A Raptor code has parameters (k, C, Ω).

In this work, we will concentrate on LT codes.

1.3.2 Systematic LT Codes

The design of systematic LT codes was introduced in [47]. A systematic LT code is an LT code for which the input bits appears as part of the output bits; these input bits are called systematic bits.

In what follows, x and w are row vectors of length k, z and y are row vectors of length N, and $\epsilon > 0$.

The systematic LT encoding requires two steps. First the computation of systematic positions $i_1, \cdots, i_k$ that can be done as follows.

Algorithm 2. *Calculation of systematic positions*
Input: LT code with parameters (k, Ω).
Output: If successful, row vectors $v_1, \cdots, v_{k(1+\epsilon)} \in \mathrm{GF}(2)^k$*, indices* $i_1, \cdots, i_k$ *between* 1 *and* $k(1+\epsilon)$*, and an invertible* $k \times k$ *matrix* G *such that* G *is the matrix formed by rows* $v_{i_1}, \cdots, v_{i_k}$.

1. *Sample* $k(1+\epsilon)$ *times independently from the distribution* Ω *on* $\mathrm{GF}(2)^k$ *to obtain* $v_1, \cdots, v_{k(1+\epsilon)}$.
2. *Let* S *be the* $k(1+\epsilon) \times k$ *matrix consisting of rows* $v_1, \cdots, v_{k(1+\epsilon)}$.
3. *Calculate rows* $i_1, \cdots, i_k$ *such that the submatrix* G *of* S *consisting of these rows is invertible, and calculate* G^{-1}.
4. *If* $i_1, \cdots, i_k$ *don't exist, goto 1.*

The calculation of rows $i_1, \cdots, i_k$ is done by removing $k\epsilon$ redundant rows in S. The author [47] suggested to use Gaussian elimination to obtain G.

The second step is to generate output bits $z_1, \cdots, z_N$ such that $z_{i_1} = x_1, \cdots, z_{i_k} = x_k$.

The following algorithm describes how to generate the output bits for a systematic LT code.

Algorithm 3. *Systematic LT encoder rule*
Input: Row vector of systematic bits $x = (x_1, \cdots, x_k)$ *and LT code with parameters* (k, Ω).
Output: Row vector of output bits $z = (z_1, \cdots, z_N)$ *such that* $z_1 = x_1, \cdots, z_k = x_k$.

1. *Use Algo. 2 to obtain* G *and calculate the intermediate bit vector* $w = (w_1, \cdots, w_k)$ *given by*
$$w = G^{-1} \cdot x.$$

2. *Generate the output bits* $z_{k+1}, z_{k+2}, \cdots, z_N$ *by applying the LT code with parameters* (k, Ω) *to the row vector* w.

Example 1. *We illustrate calculation of systematic positions and LT systematic encoding rules in Fig. 1.3, where* $k = 7$ *and* $\epsilon = \frac{4}{7}$.

Let us denote the upper white nodes (with no dashed circles) by $node_i$ *where* $i \in \mathfrak{I}_1^7$. *Let us denote the lower* 11 *nodes by* $node_{v_j}$, *where* $j \in \mathfrak{I}_1^{11}$. *A row vector* $v_j = (v_{j1}, ..., v_{j11})$ *is obtained by looking at the connection between* $node_{v_j}$ *and nodes* $node_i$ *as follows.*

$$v_{ji} = \begin{cases} 1 & \text{if there is a connection between } node_i \text{ and } node_{v_j} \\ 0 & \text{if there is no connection between } node_i \text{ and } node_{v_j} \end{cases}$$

S *is the matrix consisting of rows* $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}$, *and* G *is the matrix consisting of rows* $v_2, v_4, v_5, v_7, v_8, v_{10}, v_{11}$. *Matrices* S *and* G *are given by*

$$S = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, G = \begin{bmatrix} v_2 \\ v_4 \\ v_5 \\ v_7 \\ v_8 \\ v_{10} \\ v_{11} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

■

The systematic LT decoding is done as follows.

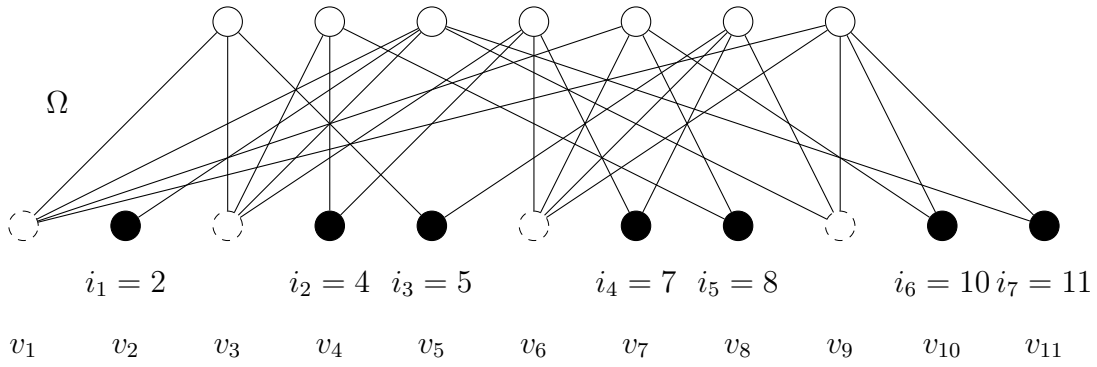


Figure 1.3: A systematic LT code: Matrices S and G

Algorithm 4. *Systematic LT decoder rule*
Input: Row vector of noisy output bits $y = (y_1, \cdots, y_N)$ *of an LT code with parameters* (k, Ω).
Output: Row vector of estimates of input bits $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. *Decode the output bits using a decoding algorithm for the original LT code to obtain the row vector of estimates of intermediate bits* $\widehat{w} = (\widehat{w_1}, \cdots, \widehat{w_k})$.
2. *Calculate* $\widehat{x} = G \cdot \widehat{w}$, *where* $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ *and* $\widehat{w} = (\widehat{w}_1, \cdots, \widehat{w}_k)$

For BEC(p) channels with parameter p, the decoder for the systematic LT code collects $N = k(1+\epsilon)$([47]) output bits and recovers the input bits with negligible error probability.

1.3.3 Sum-Product Decoding for Binary Alphabets

The LT decoder can use the Sum-Product (SP) algorithm (see [17] and [29]) to recover the input bits from the information contained in the output bits. In this part, we describe the SP algorithm for BIMSCs.

The SP algorithm is the standard decoding algorithm for graph-based codes. In the case of finite cycle-free graphs, it is finite and exact. For connected graphs with cycles, the SP-based decoding algorithm is suboptimal.

Under the SP algorithm, messages are passed along edges, and are functions of the variables incident to the edge. The messages are probabilities of the incident variable x_s and can be represented as LLRs

$$\ln \frac{\Pr[x_s = 0]}{\Pr[x_s = 1]}$$

where $s \in \mathcal{I}_1^k$.

The SP decoding algorithm proceeds in rounds. At every round $d \geq 0$,

messages are passed from input nodes to output nodes, and then from output nodes back to input nodes along the edges of a decoding graph for the given LT code.

Let us denote the input node of degree i by inb^i, the output node of degree j by $outb^j$, the message sent from input node inb^i to output node $outb^j$ at round d by $M^{(d)}_{inb^i \to outb^j}$, the message sent from output node $outb^j$ to input node inb^i at round d by $M^{(d)}_{inb^i \leftarrow outb^j}$and

$$\begin{aligned} M_{\mathcal{C}_2} &= \left(M_{\mathcal{C}_{2,1}}, \cdots, M_{\mathcal{C}_{2,N}}\right) \\ &\triangleq \left(\ln \frac{\Pr[z_1 = 0|y_1]}{\Pr[z_1 = 1|y_1]}, \cdots, \ln \frac{\Pr[z_N = 0|y_N]}{\Pr[z_N = 1|y_N]}\right) \end{aligned}$$

as the LLR information row vector of length N of the channel $\mathcal{C}_2$, where $z = (z_1, \cdots, z_N)$ is the vector sent on the channel and $y = (y_1, \cdots, y_N)$ the received vector.

For $j, s \in \mathcal{I}_1^k$ and $i, r \in \mathcal{I}_1^N$, we assume that $outb^j$ is associated to the variable y_r, inb^i is associated to the variable x_s and $M_{\mathcal{C}_{2,r_r}}$ is the LLR of the channel $\mathcal{C}_2$ for $outb^j$. In the very first round, output nodes with degree 1, $outb^1$, send their values to their unique neighbours in the set of the input nodes, which can be written as

For $r \in \mathcal{I}_1^N$, do

$$M^{(0)}_{inb^i \leftarrow outb^j} = \begin{cases} M_{\mathcal{C}_{2,r}} & \text{if the degree of } outb^j \text{ is 1,} \\ 0 & \text{if the degree of } outb^j \text{ is bigger than 1.} \end{cases}$$

For $d \geq 0$, the SP update rules (see Fig. 1.5) for the next steps are given as follows.

Algorithm 5. *Sum-Product decoding rule*
Input: Row vector of noisy output bits $y = (y_1, \cdots, y_N)$ and the corresponding graph for an LT code with parameters (k, Ω).
Output: Row vector of estimates of input bits $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. **Update rule for input nodes.**
   Set the components of the message $M^{(d)}_{inb^i \to outb^j}$ from input node inb^i to output node $outb^j$ as follows.
   For $s \in \mathcal{I}_1^k$, do
   $$M^{(d)}_{inb^i \to outb^j} = \sum_{p=1}^{i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p}}$$
   where the messages $M^{(d)}_{inb^i \leftarrow outb^{j'_1}}, \cdots, M^{(d)}_{inb^i \leftarrow outb^{j'_{i-1}}}$ are related to all output nodes $outb^{j'_p}$ adjacent to inb^i other than $outb^j$, $p \in \mathcal{I}_1^{i-1}$ and $j'_p \in \mathcal{I}_1^k$.

2. **Update rule for output nodes.**
   Set the components of the message $M^{(d+1)}_{inb^i \leftarrow outb^j}$ from output node $outb^j$ to

input node inb^i *as follows.*
For $r \in \mathfrak{I}_1^N$, *do*

$$M_{inb^i \leftarrow outb^j}^{(d+1)} = 2\text{arctanh}\left(\tanh\left(\frac{M_{\mathfrak{C}_{2,r}}}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M_{inb^{i'_l} \rightarrow outb^j}^{(d)}}{2}\right)\right)$$

where the messages $M_{inb^{i'_1} \rightarrow outb^j}^{(d)}, \cdots, M_{inb^{i'_{j-1}} \rightarrow outb^j}^{(d)}$ *are related to all input nodes* $inb^{i'_l}$ *adjacent to* $outb^j$ *other than* inb^i, $l \in \mathfrak{I}_1^{j-1}$ *and* $i'_l \in \mathfrak{I}_1^N$.

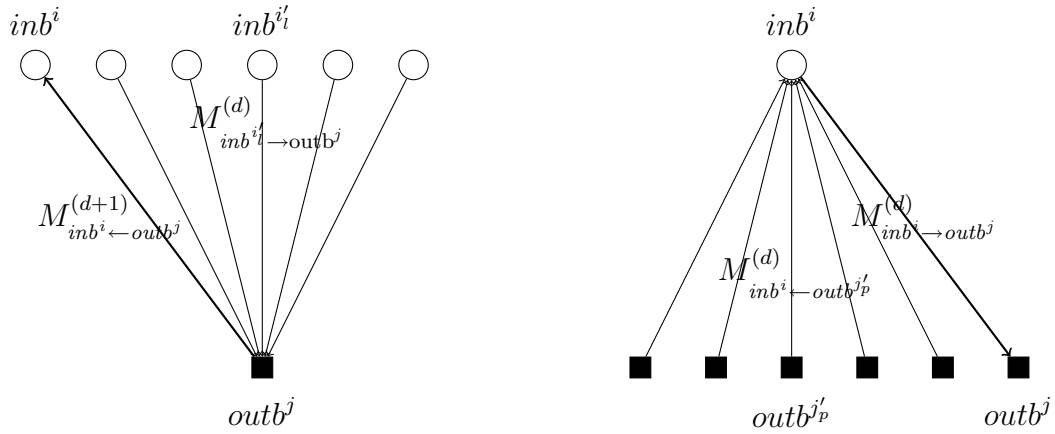


Figure 1.4: Sum-Product decoding for binary alphabets

In practical settings, the decoder stops after a fixed number of SP decoding rounds and computes at each input node inb^i a maximum a posteriori estimation of the s^{th} transmitted information bit x_s, i.e.,

$$\arg_{x_s, s \in \mathfrak{I}_1^N} \max \Big(\Pr(x_s | y_1, \cdots, y_N)\Big).$$

Let us define the function $F(x) \triangleq \frac{e^x}{1+e^x}$, where x is a real. If a real t is a message sent from (or to) an input node, then $F(t)$ $(1 - F(t)$, respectively) represents an estimate of the a posteriori probability that the corresponding code bit is 0 (1, respectively), given the LLR channel observations in the corresponding decoding graph. The decoder decides on the bit 0 (1, respectively) that maximizes $F(t)$ ($1 - F(t)$, respectively). The decision associated with x is defined as follows.

The MAP estimation of the transmitted information bit x_s associated to the input node inb^i is computed by the SP decoding as

$$\widehat{x_s} = \begin{cases} 0 & \text{if } P_0 > P_1 \\ 1 & \text{if } P_0 < P_1 \end{cases}$$

where

$$P_0 = 1 - P_1 \triangleq F\Big(\sum_{r=1}^{i} M^{(d_{end})}_{inb^i \leftarrow outb^{j'_r}}\Big)$$

the sum being over all output nodes $outb^{j'_r}$ adjacent to inb^i, $r \in \mathcal{I}_1^i$, $j'_r \in \mathcal{I}_1^k$, $i \in \mathcal{I}_1^N$ and $d_{end} > 0$ is the last round of the SP decoding.

In what follows, we generalize the binary SP algorithm to the non binary case.

1.3.4 Prerequisites for Non Binary Decoding Algorithms

Let us denote an integer $m \geq 0$, and $q = 2^m$. Given a binary LT code with parameters (k, Ω), we construct non binary LT codes over GF(q). We assume that symbols of the code are elements of GF(q). We fix a bijection between GF(q) and $\mathcal{I}_0^{q-1}$, and a bijection between GF(q) and GF(2)^m. In other words, we map every element $x \in$ GF(q) to an unique integer $a \in \mathcal{I}_0^{q-1}$. We also map each element $x \in$ GF(q) to the bit row vector of length m

$$b(x) = [b_1(x), \cdots, b_m(x)],$$

where $b_\tau(x) \in$ GF(2); $\tau \in \mathcal{I}_1^m$ refers to the τ^{th} bit-plane position of x and $b_\tau(x)$ is the bit corresponding to the τ^{th} bit-plane of x.

In what follows, each element $x \in$ GF(q) is identified with $b(x) \in$ GF(2)^m or $a \in \mathcal{I}_0^{q-1}$. $\forall x, y \in$ GF(q), it is then clear that $b(x + y) = b(x) + b(y)$,

Definition 1. *A q-dimensional probability row vector is a row vector $v = (v_0, \cdots, v_{q-1})$ of non negative real numbers, where $\sum_{h \in \mathcal{I}_0^{q-1}} v_h = 1$.*

1.3.5 Sum-Product Decoding for Non Binary Alphabets

For q-ary LT SP decoders, messages are multidimensional vectors rather than scalar values as in the standard binary LT code. Assuming that the underlying alphabet is GF(q), where $q = 2^m$ and $m > 0$, the messages that are passed along the edges of the graph are probability distributions on GF(q), and can be described by a vector of $(q-1)$ non-negative real numbers with sum at most equal to 1. The message passing algorithm performs convolutions of these probability distributions on one side of the graph, and point-wise products (and renormalization) on the other side.

The multidimensional channel information vector of a received row vector of symbols $y = (y_1, \cdots, y_N)$ is defined as

$$\begin{aligned} M_{\mathfrak{C}_q} &\triangleq (M_{\mathfrak{C}_{q,0}}, \cdots, M_{\mathfrak{C}_{q,q-1}}) \\ &= \left(\frac{\Pr[y_r|z_r = 0]}{\sum_{k=0}^{q-1} \Pr[y_r|z_r = k]}, \cdots, \frac{\Pr[y_r|z_r = q-1]}{\sum_{k=0}^{q-1} \Pr[z_r|z_r = k]}\right) \end{aligned}$$

where $r \in \mathcal{I}_1^N$.

As for the binary case, the q-ary LT SP decoding algorithm proceeds in rounds. At every round $d > 0$, we denote the input node of degree i by inb^i, the output node of degree j by $outb^j$, the multidimensional message sent from inb^i to $outb^j$ at round d by $M^{(d)}_{inb^i \to outb^j}$ and the multidimensional message sent from $outb^j$ to inb^i at round d by $M^{(d)}_{inb^i \leftarrow j}$:

$$M^{(d)}_{inb^i \to outb^j} \triangleq (M^{(d)}_{inb^i \to outb^j,0}, \cdots, M^{(d)}_{inb^i \to outb^j,q-1})$$

$$M^{(d)}_{inb^i \leftarrow outb^j} \triangleq (M^{(d)}_{inb^i \leftarrow outb^j,0}, \cdots, M^{(d)}_{inb^i \leftarrow outb^j,q-1})$$

The decoder attempts to recover input symbols $x_1, \cdots, x_k$. The decoding consists of the iterative sending of messages from output nodes to input nodes and from input nodes to output nodes as follows.

For $j, s \in \mathcal{I}_1^k$ and $i, r \in \mathcal{I}_1^N$, we assume that $outb^j$ is associated to the variable y_r, inb^i is associated to the variable x_s and $M_{\mathcal{C}_q}$ is the LLR of the channel $\mathcal{C}_q$ for $outb^j$. In the very first round, output nodes with degree 1, $outb^1$, send their values to their unique neighbours in the set of the input nodes, which can be written as

For $r \in \mathcal{I}_1^N$ set

$$M^{(d)}_{inb^i \leftarrow outb^j} = \begin{cases} M_{\mathcal{C}_q} & \text{if the degree of } outb^j \text{ is 1} \\ 0 & \text{if the degree of } outb^j \text{ is bigger than 1} \end{cases}$$

For $d \geq 0$, the SP update rules for the next steps are given as follows.

Algorithm 6. *Sum-Product decoding rule*
Input: Row vector noisy output symbols $y = (y_1, \cdots, y_N)$ and the corresponding graph for an LT code with parameters (k, Ω).
Output: Row vector of estimates of input symbols $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. *Update rule for input nodes.*
   Set the components of the message $M^{(d)}_{inb^i \to outb^j}$ from input node i of degree d_i to output node j as follows.
   For $s \in \mathcal{I}_1^k$ and for $h \in \mathcal{I}_0^{q-1}$ set

$$M^{(d)}_{inb^i \to outb^j,h} = \frac{\prod_{p=1}^{i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p},h}}{\sum_{h' \in \mathcal{I}_0^{q-1}} \prod_{p=1}^{d_i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p},h'}}$$

   where the messages $M^{(d)}_{inb^i \leftarrow outb^{j'_1},h}, \cdots, M^{(d)}_{inb^i \leftarrow outb^{j'_{i-1}},h}$ are related to all output nodes $outb^{j'_p}$ adjacent to inb^i other than $outb^j$, $p \in \mathcal{I}_1^{i-1}$ and $j'_p \in \mathcal{I}_1^k$.

2. *Update rule for output nodes.*
   Set the components of the message $M^{(d)}_{inb^i \leftarrow outb^j}$ from output node j of

degree d_j to input node i as follows.
For $r \in \mathfrak{I}_1^N$ and for $h \in \mathfrak{I}_0^{q-1}$ set

$$M^{(d+1)}_{inb^i \leftarrow outb^j,h} = M_{\mathfrak{C}_{q,r}} \sum_{\substack{a_1,\cdots,a_{j-1} \in \mathrm{GF}(q),\\ y_j + \sum_{l=1}^{d_j-1} a_l = h}} \prod_{l=1}^{j-1} M^{(d)}_{inb^{i'_l} \rightarrow outb^j,a_l}$$

where the messages $M^{(d)}_{inb^{i'_1} \rightarrow outb^j,a_1}, \cdots, M^{(d)}_{inb^{i'_{j-1}} \rightarrow outb^j,a_{j-1}}$ are related to all input nodes $inb^{i'_l}$ adjacent to $outb^j$ other than inb^i, $a_l \in \mathfrak{I}_0^{q-1}$, $l \in \mathfrak{I}_1^{j-1}$ and $i'_l \in \mathfrak{I}_1^N$.

In practical settings, the decoder stops after a fixed number of SP decoding rounds and computes at each input node inb^i a maximum a posteriori estimate of the s^{th} transmitted information symbol x_s, where $s \in \mathfrak{I}_1^k$.

The maximum a posteriori estimate of the transmitted information symbol x_s associated to input node inb^i is computed by the SP decoding as

$$\widehat{x}_s = \arg_{h \in \mathfrak{I}_0^{q-1}} \max \left(\frac{\prod_{p=1}^{i-1} M^{(d_{end,q})}_{inb^i \leftarrow outb^{j'_p},h}}{\sum_{h' \in \mathfrak{I}_0^{q-1}} \prod_{p=1}^{d_i-1} M^{(d_{end,q})}_{inb^i \leftarrow outb^{j'_p},h'}} \right)$$

where the messages $M^{(d)}_{inb^i \leftarrow outb^{j'_1},h}, \cdots, M^{(d)}_{inb^i \leftarrow outb^{j'_{i-1}},h}$ are related to all output nodes $outb^{j'_p}$ adjacent to inb^i,$h \in \mathfrak{I}_0^{q-1}$, $j'_p \in \mathfrak{I}_1^k$ and $d_{end,q}$ is the last round of the SP decoding.

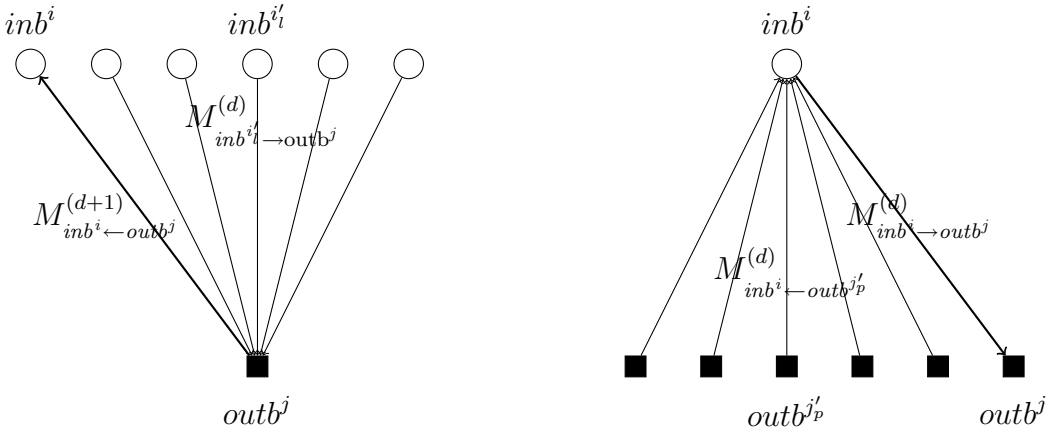


Figure 1.5: Sum-Product decoding for q-ary alphabets

1.3.6 Hadamard-Based SP Decoding for Non Binary Alphabets

Definition 2. *The Hadamard-Walsh matrix of size $q = 2^m$ is the $2^m \times 2^m$ matrix recursively defined by*

$$H_m = \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix}$$

where $H_0 = 1$ and $m > 0$.

Example 2. *Matrices H_1 and H_2 are given by*

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

■

The Hadamard-Walsh matrix H_m has the following properties :

- $H_m = {H_m}^\top$, where ${H_m}^\top$ is the transpose of the matrix H_m
- The inverse of H_m is $\frac{1}{q}H_m$.

Definition 3. *Let $v = (v_0, \cdots, v_{q-1})$ and $w = (w_0, \cdots, w_{q-1})$ be two q-dimensional row vectors of real numbers. The multiplication of v and w performed component-wise is the vector $v * w = (u_0, \cdots, u_{q-1})$ defined as*

$$u = (v_0 w_0, \cdots, v_{q-1} w_{q-1})$$

Definition 4. *Let $v = (v_0, \cdots, v_{q-1})$ be a $1 \times q$ vector. The Hadamard transform of v is the vector*

$$vH_m.$$

Theorem 1. *Let $d > 0$, and suppose that $X_1, \cdots, X_d$ are d independent random variables defined over $\mathrm{GF}(q)$. For $i \in \mathcal{I}_1^d$, we define by $\Pr(X_i)$, the distribution of X_i:*

$$\Pr(X_i) \triangleq \Big(\Pr[X_i = 0], \cdots, \Pr[X_i = q-1]\Big).$$

Then, the probability density of $Y = X_1 + \cdots + X_d$ denoted by $\Pr(Y)$, is given by

$$\begin{aligned} \Pr(Y) &\triangleq \Big(\Pr[Y = 0], \cdots, \Pr[Y = q-1]\Big) \\ &= \frac{1}{q}\Big(\prod_{i=1}^{d}\Big(\Pr(X_i)H_m\Big)\Big)H_m \end{aligned} \tag{1.1}$$

where $\Pr(x_i)H_m$ is the Hadamard transform vector of $\Pr(x_i)$ and the multiplication of the Hadamard transform vectors is performed component-wise.

Proof: See, e.g., in [30].

■

In the decoding algorithm we previously presented, at each output node $outb^j$ the update rule for input nodes step performs convolutions of probability distributions at cost proportional to jq^{j-1}. These convolutions can be performed at cost proportional to $jq\log_2(q)$ using a Hadamard-Walsh transform [36, 39]. Using the formula (1.1), one can change the update rule for output node into a product node in the decoding graph and then obtain another version of SP decoding, called the Hadamard-based SP, which is given as follows.

In the very first round (see the formula before Algo.6), output nodes with degree 1 send the values to whatever is coming from the channel to their unique neighbours in the set of the input nodes.

Algorithm 7. *Hadamard-based SP decoding rule*
Input: Row vector noisy of output symbols $y = (y_1, \cdots, y_N)$ *and the corresponding graph for an LT code with parameters* (k, Ω).
Output: Row vector of estimates of input symbols $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. *Update rule for input nodes.*
   Set the components of the message $M^{(d)}_{inb^i \to outb^j}$ *from input node* i *of degree* d_i *to output node* j *as follows.*
   For $s \in \mathfrak{I}_1^k$ *and for* $h \in \mathfrak{I}_0^{q-1}$ *set*

$$M^{(d)}_{inb^i \to outb^j,h} = \frac{\prod_{p=1}^{i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p},h}}{\sum_{h' \in \mathfrak{I}_0^{q-1}} \prod_{p=1}^{d_i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p},h'}}$$

   where the messages $M^{(d)}_{inb^i \leftarrow outb^{j'_1},h}, \cdots, M^{(d)}_{inb^i \leftarrow outb^{j'_{i-1}},h}$ *are related to all output nodes* $outb^{j'_p}$ *adjacent to* inb^i *other than* $outb^j$, $p \in \mathfrak{I}_1^{i-1}$ *and* $j'_p \in \mathfrak{I}_1^k$.

2. *Update rule for output nodes.*
   Set the components of the message $M^{(d)}_{inb^i \leftarrow outb^j}$ *from output node* j *of degree* d_j *to input node* i *as follows.*
   For $r \in \mathfrak{I}_1^N$ *and for* $h \in \mathfrak{I}_0^{q-1}$ *set*

$$M^{(d+1)}_{inb^i \leftarrow outb^j} = \frac{1}{q}\left(\left(M_{\mathcal{C}_q} H_m\right) * \prod_{l=1}^{j-1}\left(M^{(d)}_{inb^{i'_l} \to outb^j} H_m\right)\right) H_m$$

   where the q*-dimensional messages* $M^{(d)}_{inb^{i'_1} \to outb^j}, \cdots, M^{(d)}_{inb^{i'_{j-1}} \to outb^j}$ *are related to all input nodes* $inb^{i'_l}$ *adjacent to* $outb^j$ *other than* inb^i, $l \in \mathfrak{I}_1^{j-1}$, $i'_l \in \mathfrak{I}_1^N$, *and the product of the Hadamard transform vectors is performed component-wise.*

Binary LT Codes for Nonbinary Channels

2

In Chapter 1, we gave descriptions of binary Sum-Product (SP) decoding algorithms that are used in the decoding process of LT codes, respectively for binary alphabets and non binary alphabets. In this chapter, we address the problem of decoding non binary LT codes over finite fields of large order. We propose a new decoding algorithm for non binary channels that provides a trade-off between complexity and decoding capability in contrast to the existing well-known non-binary SP algorithm. An introduction is given in Section 2.1. In Section 2.2, we introduce our new decoding algorithm for LT codes used for the transmission over non-binary channels using linear forms that combine binary SP and Maximum Likelihood (ML) decoding algorithms. Simulation results showing the performance of the new decoding algorithms as compared to the non binary SP and multilevel decoding algorithms are finally provided in 2.3.

2.1 Introduction

The success of iterative decoding algorithms for graph-based codes on binary memoryless channels has for some time invited the question whether it is possible to generalize binary decoding algorithms, tools for their design and the analysis to non binary alphabets. Such generalizations are needed in a variety of applications, such as communication applications that require higher modulation than binary and applications to compression of files where the alphabet elements are naturally bytes.

It is relatively straightforward to generalize the binary SP algorithm to the non binary case as we saw in Section 1.3.5 ([11]).

However, despite the improvements reported in [12], the problem of decoding graphical codes over GF(q) remains a difficult computational problem,

particularly when the field size q increases. SP based methods are rendered completely ineffective when the field size q is very large, as is for example the case in packet based transmissions where q can have a value of 2^{8192} or more. In these cases, under the assumption that the transmission is performed on the q-ary symmetric channel (q-SC), several algorithms have been provided [32, 48, 27]. However, these algorithms are ineffective when q is not too large, say in the range of a few hundreds to a few thousands.

In this chapter, we are going to ask a slightly different question: to what extent can we use binary codes on non-binary channels and how can we design efficient decoding algorithms? It is quite clear that if the q-ary channel introduces independent errors on the bits of a transmitted element, then the error correction capability of binary codes is similar to that of binary codes on binary channels. However, if the channel does not introduce independent errors at the bit-level (something that is common for q-ary channels), then binary codes could have an advantage. We introduce a class of decoding algorithms for binary LT codes used for transmission over q-ary channels. The algorithms provide a trade-off between complexity and decoding capability. Whereas the running time of the q-ary SP algorithm is $m2^m$ times that of its binary counterpart (see the paragraph after Thoerem 1 in Chapter 1), in our case the complexity factor can be chosen between m and 2^m, depending on the error-correction capability required. As such, the running time can be much better than the q-ary SP algorithm. The main idea behind our algorithm is to apply an appropriate set of GF(2)-linear forms of GF(q) to the LT code to obtain a set of binary codes which can be decoded independently in parallel. After a prescribed number of iterations, for each of the input symbols of the LT code and each of the linear forms, an estimate is obtained on the probability that this linear form applied to the input symbol is zero. By gathering these probabilities and performing a ML decoding on a suitable code of very small block-length, we are able to obtain a good estimate of the value of the input symbol.

In the following, let X be a random variable over GF(q), and

$$p = (p_0, \cdots, p_{q-1})$$

denote a probability mass function on GF(q) defining the distribution of a random variable X, i.e,

$$\Pr[X = x] = p_x$$

for each $x \in \mathrm{GF}(q)$. Furthermore, let $\varphi : \mathrm{GF}(q) \to \mathrm{GF}(2)$, a linear form. Then φ can be reprensented as

$$\varphi(x) = \varphi\Big([b_1(x), \cdots, b_m(x)]\Big) = \varphi^{(1)} b_1(x) + \cdots + \varphi^{(m)} b_m(x)$$

where $x \in \mathrm{GF}(q)$, $\varphi^{(\tau)} \in \mathrm{GF}(2)$ and $\tau \in \mathfrak{I}_1^m$. We identify the linear form φ with the bit row vector of length m, $(\varphi^{(1)}, \cdots, \varphi^{(m)})$. A linear form φ identified by

$(\varphi^{(1)}, \cdots, \varphi^{(m)})$ is called non zero linear form if there exists $i \in \mathfrak{I}_1^m$ such that $\varphi^{(i)} \neq 0$.

The quantity $\Pr[\varphi(X) = 0]$ is defined as the probability that $\varphi(X) = 0$ is zero for all non zero linear forms on $\mathrm{GF}(q)$. We remark that this probability equals

$$\Pr[\varphi(X) = 0] = \sum_{h \in \mathrm{GF}(q)} \Pr[X = h] \mathbf{1}\Big(\varphi(X) = 0\Big).$$

Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a set of n linear forms from $\mathrm{GF}(q)$ to $\mathrm{GF}(2)$. Assume that $\dim\langle\Phi\rangle = m$. The code C_Φ defined by

$$C_\Phi = \{(\varphi_1(x), \ldots, \varphi_n(x)) \mid x \in \mathrm{GF}(q)\}$$

is a linear code of block-length n. Since we have assumed that $\dim\langle\Phi\rangle = m$, it follows that $\dim(C_\Phi) = m$. Hence, C_Φ is an $[n, m]_2$-code. An $m \times n$ generator matrix $G(C_\Phi)$ for the $[n, m]_2$-code C_Φ is given by

$$G(C_\Phi) = \begin{bmatrix} \varphi_1^{(1)} & \cdots & \varphi_n^{(1)} \\ \vdots & \vdots & \vdots \\ \varphi_1^{(m)} & \cdots & \varphi_n^{(m)} \end{bmatrix},$$

if each linear form φ_v is identified with the bit row vector $(\varphi_v^{(1)}, \cdots, \varphi_v^{(m)})$. A special case is obtained when $\varphi_1, \ldots, \varphi_n$ are all the nonzero linear forms of $\mathrm{GF}(q)$. In this case, C_ϕ is the Hadamard code of dimension m and block-length $q - 1$.
Another special case is obtained with only fundamental linear forms, i.e.,

$$\varphi_i^{(j)} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

where $i, j \in \mathfrak{I}_1^m$. In this case, C_ϕ is a $[m, m]_2$-code.

Example 3. $\mathrm{GF}(q) = \mathrm{GF}(2^2)$.
Let $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$ *where*

$$G(C_\Phi) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

C_Φ *is a* $[3, 2]_2$*-code called the Hadamard-code of dimension* 2.

■

Example 4. $\mathrm{GF}(q) = \mathrm{GF}(2^4)$.
Let $\Phi = \{\varphi_1, \varphi_2, \varphi_3\, \varphi_4, \varphi_5\, \varphi_6, \varphi_7\}$ *where*

$$G(C_\Phi) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

C_Φ *is a* $[7, 4]_2$*-code called the Hamming-code.* ■

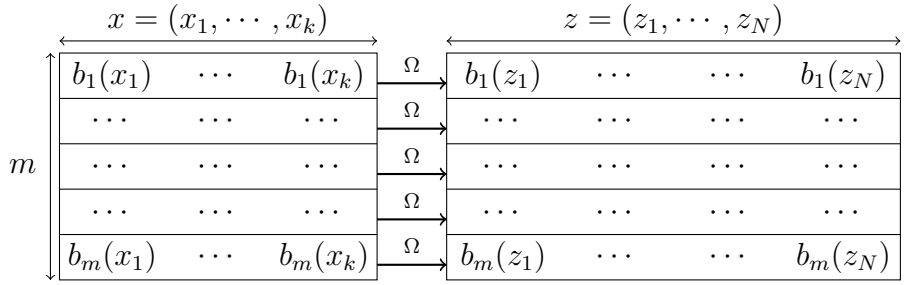


Figure 2.1: GF(q)-LT Codes: Encoding

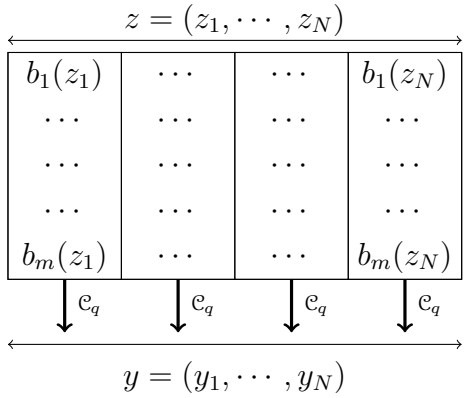


Figure 2.2: GF(q)-LT Codes: Transmission

Example 5. $\mathrm{GF}(q) = \mathrm{GF}(2^8)$.
Let $\Phi = \{\varphi_1, \varphi_2, \varphi_3\, \varphi_4, \varphi_5, \varphi_6, \varphi_7, \varphi_8\}$ *where*

$$G(C_\Phi) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

C_Φ *is a* $[8, 8]_2$*-code obtained with only fundamental linear forms.* ■

2.2 SP Decoding Algorithm using Linear Forms

Let $\mathfrak{L}$ be a binary LT-code with parameters (k, Ω). A row vector $x = (x_1, \cdots, x_k) \in \mathrm{GF}(q)^k$ can be encoded with $\mathfrak{L}$ into a row vector $z = (z_1, \cdots, z_N)$, as can be

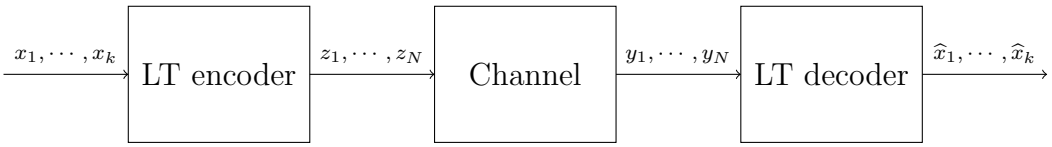


Figure 2.3: LT communication system for q-ary alphabets.

seen in Fig. 2.1. These output symbols are sent over a q-ary channel $\mathcal{C}_q$, and are received as symbols $y = (y_1, \cdots, y_N)$. The transmission (see Fig. 2.2) is done symbol-by-symbol. The value of N does not need to be known in advance. The row vector $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ denotes an estimate of x obtained after the decoding. The complete LT communication system is illustrated in Fig. 2.3. We will map the symbols of GF(q) to bit-vectors of length n using a linear error-correcting code C_Φ of dimension m and block-length n (as can be seen in Fig. 2.4).

2.2.1 Overview of the algorithm

The main observation is that for a fixed position $r \in \mathcal{I}_1^N$, the bits of all the symbols of the codeword in that position are elements of the specific chosen LT code. Once the codeword $y = (y_1, \cdots, y_N)$ is received (with soft information about each coordinate position), then we apply the linear forms φ corresponding to the code C_Φ, and consider the vector $(\varphi(y_1), \ldots, \varphi(y_N))$. This vector is an element of the chosen LT code, and can be decoded using binary decoding algorithms. By repeating the decoding process for all the linear forms corresponding to the code C_Φ , we obtain for every position $r \in \mathcal{I}_1^N$ an estimate on the codeword (in C_Φ) corresponding to y_r. The ML decoding in C_Φ will give for each such position a candidate codeword, which can be decoded to an element of GF(q). Row vectors $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ representing estimates of $x = (x_1, \cdots, x_k)$ are obtained after the decoding.

This algorithm has the advantage that its running time scales with the length n of C_Φ; in the worst case, where C_Φ is the Hadamard code, the running time scales with q and not with $q \log_2(q)$, as is the case for the q-ary belief-propagation algorithm. Moreover, we can choose the code C_Φ as a function of the application, thereby reducing the complexity of the decoding process at the expense of a worse error-correction capability.

2.2.2 Detailed description of the algorithm

Taking binary linear combinations of the GF(q)-elements yields binary symbols defined over GF(q). Every output symbol of the code is a binary linear combination of some of the input symbols. By applying the same binary linear combination to the GF(q)-input symbols, we obtain GF(q)-output symbols of

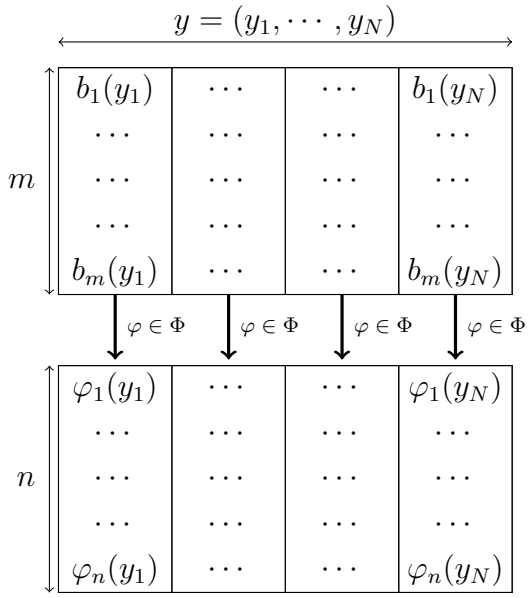


Figure 2.4: $\mathrm{GF}(q)$-LT Codes: Linear forms

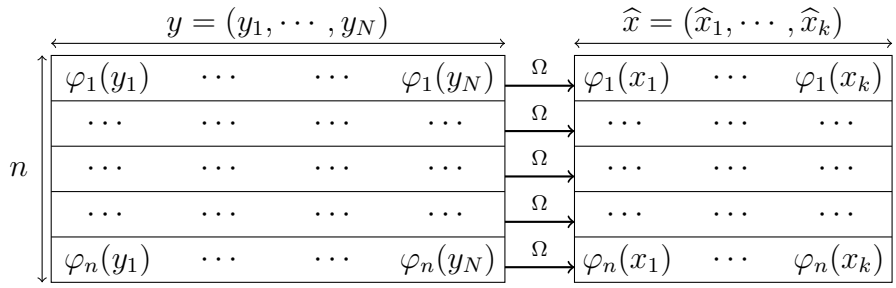


Figure 2.5: $\mathrm{GF}(q)$-LT Codes: Decoding using linear forms

this LT code. This is the special type of non-binary LT codes that we study in this section. The following remark is an immediate consequence of the fact that the code is defined by binary linear forms.

Remark 1. *Let $x_1, \cdots, x_k$ be k q-ary source symbols of an LT code with parameters $(k, \Omega(x))$, and let $y_1, \cdots, y_N$ be N q-ary output symbols received. Let G denote the corresponding decoding graph. Then for every linear form $\varphi\colon \mathrm{GF}(q) \to \mathrm{GF}(2)$ the graph G is the decoding graph between the input bits $\varphi(x_1), \ldots, \varphi(x_k)$ and the output bits $\varphi(y_1), \ldots, \varphi(y_N)$, as illustrated in Fig. 2.5.*

This remark is the key observation leading to the decoding algorithms described in this section. We will call the code with the source symbols $\varphi(x_1), \cdots, \varphi(x_k)$ the marginalized LT code with respect to φ.

We will use the code C_Φ and Remark 1 to develop a decoding algorithm for non-binary LT codes as described above.

Suppose that $\Pr(z_r \mid y_r)$ is the probability of having sent $z_r \in \mathrm{GF}(q)$ provided that $y_r \in \mathrm{GF}(q)$ is received, where $r \in \mathcal{I}_1^N$. Suppose now that y_r is received, and let φ be a linear form on $\mathrm{GF}(q)$. Then, the posterior probability that $\varphi(z_j)$ is zero given that y_r is received is a marginal probability equal to

$$\Pr[\varphi(z_r) = 0 \mid y_r] = \sum_{\substack{u \in \mathrm{GF}(q) \\ \varphi(u)=0}} \Pr(u \mid y_r), \tag{2.1}$$

where the marginalization is obtained by summing over all probabilities of having sent symbols $u \in \mathrm{GF}(q)$ provided that y_r is received and $\varphi(u)$ is zero.

With these preparations, we are ready to describe our decoding algorithms for non-binary LT codes. The set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ of linear forms is assumed to be fixed throughout the algorithm.

For $i \in \mathcal{I}_1^n$ and $s \in \mathcal{I}_1^k$, we define G_i as the decoding graph between input bits $\varphi_i(x_1), \ldots, \varphi_i(x_k)$ and output bits $\varphi_i(y_1), \ldots, \varphi_i(y_N)$.
Let us define $S_{\varphi_i(x_s)}$ as the log-likelihood ratio expressing a soft belief in the current value of $\varphi_i(x_s)$ associated to the decoding graph G_i. The a-priori probability that $\varphi_i(x_s)$ is zero is given by

$$\Pr[\varphi_i(x_s) = 0] \triangleq \frac{\exp^{(S_{\varphi_i(x_s)})}}{1 + \exp^{(S_{\varphi_i(x_s)})}} \tag{2.2}$$

The n graphs $G_1, \cdots, G_n$, built with the same degree distribution Ω, have the same graphical representation, i.e, connections between edges and nodes are identical.

Our algorithms proceed in two main steps: the SP step and the ML step. In what follows, we will describe the first variant of our decoding algorithms.

Algorithm 8. *Linear Forms based SP and ML decoding rules - First variant*
Input: Row vector noisy output symbols $y = (y_1, \cdots, y_N)$ and the corresponding graph for an LT code with parameters (k, Ω).
Output: Row vector of estimates of input symbols $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. *Sum-Product Step :*
   For $i \in \mathcal{I}_1^n$ and $r \in \mathcal{I}_1^N$,

   a) *Initialization:*
   We initialize the value of each output symbol y_r with

$$\ln\left(\frac{p_{i,r}}{1 - p_{i,r}}\right)$$

   where $p_{i,r} \triangleq \Pr[\varphi_i(z_r) = 0 \mid y_r]$ is calculated according to (2.1) and is associated to decoding graph G_i.

b) Message passing:
Then, at every step of the algorithm, we perform independently for every coordinate position i associated with the respective decoding graph G_i, an update of messages according to the binary SP algorithm.

2. *Maximum Likelihood Step:*
After a prescribed number of SP rounds, for every linear form $\varphi_i \in \Phi$ and every input symbol $x_s \in \mathrm{GF}(q)$, we calculate $\Pr[\varphi_i(x_s) = 0]$, the a-priori probability that $\varphi_i(x_s)$ is zero, according to (2.2).
Next, we calculate for every element $c = (c_1, \ldots, c_n) \in C_\Phi$ the quantity

$$\Pr[x_s = c] = \prod_{i=1}^{n} \Pr[\varphi_i(x_s) = 0]^{(1-c_i)} \Pr[\varphi_i(x_s) = 1]^{c_i}.$$

where $i \in \mathfrak{I}_1^n$ and $s \in \mathfrak{I}_1^k$.
Next, we pick the codeword c for which this value is largest, and finally we pick for x the element $\alpha \in \mathrm{GF}(q)$ whose image is the codeword c.

The algorithm stops after κ_1 rounds of the message passing step, where $\kappa_1 > 0$ is a design parameter.

This algorithm can be extended in the following manner. At each step, when the input symbols have gathered information about their values, these values can be used to update the belief of the input symbols in their value using an ML-decoding on the code C_Φ. A relaxation of this method would do the inner ML-decoding every κ_2 rounds of the message passing step, where $\kappa_2 > 0$ is a design parameter which provides yet another trade-off between the running time and the correction capability of the decoder. This new algorithm proceeds as follows.

Algorithm 9. *Linear Forms based SP and ML decoding rules - Second variant*
Input: Row vector noisy output symbols $y = (y_1, \cdots, y_N)$ and the corresponding graph for an LT code with parameters (k, Ω).
Output: Row vector of estimates of input symbols $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. *Sum-Product Step :*
For $i \in \mathfrak{I}_1^n$ and $r \in \mathfrak{I}_1^N$,

a) Initialization:
We initialize the value of each output symbol y_r with the value

$$\ln\left(\frac{p_{i,r}}{1 - p_{i,r}}\right)$$

where $p_{i,r} \triangleq \Pr[\varphi_i(z_r) = 0 \mid y_r]$ is calculated according to (2.1) and is associated to decoding graph G_i.

b) Message passing:
Then, at every step of the algorithm, we perform independently for every coordinate position i associated with the respective decoding graph G_i, an update of messages according to the binary SP algorithm.

2. *Maximum Likelihood Step :*
After a prescribed number of SP rounds, for every linear form $\varphi_i \in \Phi$ and every input symbol $x_s \in \mathrm{GF}(q)$, we calculate $\Pr[\varphi_i(x_s) = 0]$, the a-priori probability that $\varphi_i(x_s)$ is zero, according to (2.2).
Next, we calculate for every element $c = (c_1, \ldots, c_n) \in C_\Phi$ the quantity

$$\Pr[x_s = c] = \prod_{i=1}^{n} \Pr[\varphi_i(x_s) = 0]^{(1-c_i)} \Pr[\varphi_i(x_s) = 1]^{c_i}.$$

where $i \in \mathcal{I}_1^n$ and $s \in \mathcal{I}_1^k$.
Next, we pick the codeword c for which this value is largest, and finally we pick for x the element $\alpha \in \mathrm{GF}(q)$ whose image is the codeword c.

3. *We go to the SP step or stop the process after a prescribed number of iterations.*

In these algorithms, the formula we use to pick the codeword c is equal to the probability that $x_s = c$, provided that the events $\varphi_i(x_s) = 0$ are independent. In our algorithm, this may not be the case, since the a-priori probabilities $\Pr[\varphi_i(x_s) = 0]$ for different φ_i are correlated (they come from decoding on identical decoding graphs). However, our experiments reported below suggest that the correlations are weak.

2.2.3 Linear Forms and Hadamard-Based SP

In this section, we show how the Hadamard-based SP can be related to the marginalization to all linear forms.

Definition 5. *Let T_m be a linear transformation of size $2^m \times 2^m$ recursively defined by the matrix*

$$\frac{1}{2}(H_m + J_m)$$

where J_m is the $2^m \times 2^m$ matrix in which all entries are 1.

The backward transform T_m^{-1} is the inverse of the forward transform T_m. Both can be implemented in a recursive fashion. Computation costs are $O(m2^m)$.

Theorem 2. *A probability density on $\mathrm{GF}(2^m)$ is uniquely determined by the marginalization to all non zero linear forms defined from $\mathrm{GF}(2^m)$ to $\mathrm{GF}(2)$.*

Proof: We first show by simple induction reasoning that a marginalization to all linear forms can be computed by the forward transform T_m. Let us prove that for $m = 1$ (meaning that $q = 2$), a marginalization to all linear forms can be computed by the forward transform T_1. The distribution of x is defined as

$$\Pr(x) \quad = \quad (p_0, p_1).$$

The transform T_1 is given by

$$T_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

By applying the transform T_1 to $\Pr(x)$, we obtain

$$\begin{pmatrix} p_0 + p_1 \\ p_0 \end{pmatrix}.$$

Let $\Phi = \{(\varphi_1(x)\}$, which contains the unique non zero linear form from GF(2) to GF(2), the marginalization of the probability distribution $\Pr(x)$ to $\varphi_1(x)$ is given by

$$\Pr[\varphi_1(x) = 0] = p_0,$$

where $\varphi_1(x) = x_1$.

Now assume, by induction hypothesis, that a marginalization to all linear forms can be computed by the linear transform T_{m-1} over the field $\mathrm{GF}(2^{m-1})$. Let us show that we can compute a marginalization to all linear forms by the forward transform T_m over the field $\mathrm{GF}(2^m)$.
The distribution of $x \in \mathrm{GF}(2^m)$ is given by

$$\begin{aligned} \Pr(x) &= (p_0, \cdots, p_{2^{m-1}-1}, p_{2^{m-1}}, \cdots, p_{2^m-1}) \\ &= [\Pr(A), \Pr(B)] \end{aligned}$$

where $\Pr(A) \triangleq (p_0, \cdots, p_{2^{m-1}-1})$
and $\Pr(B) \triangleq (p_{2^{m-1}}, \cdots, p_{2^m-1})$

By applying the transform T_m to [Pr(A) , Pr(B)], we obtain a column vector C_m defined by

$$\begin{pmatrix} \Pr(A)T_{m-1} + \Pr(B)T_{m-1} \\ \Pr(A)T_{m-1} + \Pr(B)[-T_{m-1} + J_{m-1}] \end{pmatrix}.$$

By using the induction hypothesis $\Pr(A)T_{m-1}$+$\Pr(B)T_{m-1}$ produces the marginalization to $2^{m-1}-1$ linear forms related to the field $\mathrm{GF}(2^{m-1})$ and $\Pr(A)T_{m-1}+\Pr(B)(-T_{m-1}+J_{m-1})$ produces the marginalization to other 2^{m-1} linear forms built from the previous $2^{m-1}-1$ linear forms with the linear form representing the augmenting bit-plane representation of elements over the field $\mathrm{GF}(2^m)$ in comparison to elements over $\mathrm{GF}(2^{m-1})$.We remark that C_m is the vector of

marginalizations.

The transform T_m is full rank since it can be row reduced to

$$T_m = \begin{pmatrix} T_{m-1} & T_{m-1} \\ 0 & -2\,T_{m-1} + J_m \end{pmatrix}$$

and $-2\,T_{m-1} + 1$ is a negative Hadamard matrix which is full rank. Thus, given a marginalization to all non zero linear forms, we can compute a probability density on GF(2^m) using the backward transform T_m^{-1}, which ends the proof. ■

The consequence of this theorem is as follows. Let us consider an output symbol node of an LT code defined on GF(2^m). The idea is that for a particular output message (probability density), we do not find it directly, but first find the marginalization to all non-zero linear forms using the transform T_m. A backward transform T_m^{-1} gives the actual probability density we are looking for. To decode an ouput symbol, we can perform the following optimal decoding.

- We first apply the forward transform T_m (similar to the Hadamard transform) of the incoming probability density. This transform marginalizes to all non zero linear forms, i.e., C_Φ is only the Hadamard-code of dimension m. Then, we can just decode as if we had a set of 2^m parallel binary codes.
- Finally we perform the backward transform T_m^{-1} to the results of independent binary decoding previously obtained to find the output probability density.

2.2.4 Complexity of the SP decoding using linear forms

The running time of these algorithms can be easily assessed. The two variants (Algo. 9 and Algo. 10) of our decoding algorithms have a time and space complexity of the same order. In what follows, we derive a complexity analysis only for the the first variant.

Since at each iteration of the SP step n values are updated, the running time will be proportional to the number of edges in each decoding graph G_i with $i \in \mathcal{J}_1^n$. The running time of the ML-part of the decoding process depends on the algorithm used. The straightforward implementation of $\Pr[x = c]$ for all $c \in C_\Phi$ uses $\leq n2^m$ operations on real numbers: for every vector c we need to calculate a product of at most n terms. A more accurate estimate of the number of operations is $\sum_{c \in C} \mathrm{wgt}(c)$, where $\mathrm{wgt}(c)$ is the Hamming weight of c; hence, if C_Φ contains the all-one codeword, the running time of this algorithm will be at most $n2^m/2$.

A more efficient algorithm, along the ideas used in Theorem 2, uses the Hadamard-Walsh transform. Let $v \in \mathbb{R}^{2^m}$ be the row vector defined as $v = (\lambda_{\varphi_1}, \cdots, \lambda_{\varphi_{2^m}})$ where

$$\lambda_{\varphi_i} = \begin{cases} \ln(\Pr[\varphi_i(x) = 1]/\Pr[\varphi_i(x) = 0]) & \text{if } \varphi_i \in \Phi \\ 0 & \text{if } \varphi_i \notin \Phi \end{cases}$$

The vector v can be computed using Theorem 2, and hence the q-ary vector $(\prod_{i=1}^{n}(\Pr[\varphi_i(x) = 1]/\Pr[\varphi_i(x) = 0]) \mid x \in \mathrm{GF}(q))$, which will eventually give the values $\Pr[x = c]$ for all $c \in C_\Phi$. The cost of this approach is $O(m2^m)$, regardless of n. Hence, for $n > 2m$, one may want to use this method instead of the former.

In total, the running time of the decoder is $O(Nanl + m2^m)$, where l is the number of iterations of the SP , a is the average output degree of the LT code. For N large, the running time is dominated by $O(Nanl)$.

Depending on how we choose C_Φ, the running time can be anywhere between $O(Nam)$ and $O(Na2^m)$. Even in the worst case of $O(Na2^m)$ the running time is by a factor of m better than the running time of the SP algorithm over $\mathrm{GF}(q)$ (which is $O(Nam2^m)$).

In contrast to the q-ary SP algorithm, the new algorithm offers a tradeoff between the running time and the decoding capability.

2.3 Performance Comparison

In this section, we present simulation results of the proposed LT-code of block length N over $\mathrm{GF}(2^m)$, whose parameters are (k, Ω).

$Algo^1_{\kappa_1}$ refers to the first variant of our decoding algorithm where the ML step is performed once after κ_1 rounds of the message passing of the SP step. $Algo^2_{\kappa_2}$ refers to the second variant of our decoding algorithm where the ML is performed after every κ_2 rounds of the message passing of the SP step until convergence.

We also compare the performance of our algorithms to the non binary SP [11] and a multilevel decoding algorithm in terms of symbol error rate (SER). HSP_{κ_3} refers to the Hadamard-based SP decoding algorithm, performed during κ_3 rounds. MSP_{κ_4} refers to a variant of multilevel-based SP decoding algorithm (which will be described below), performed during κ_4 rounds.
Assume the symbol z has been sent provided that y is received, where $z, y \in \mathrm{GF}(2^m)$. Let $B(z) = (b_1(z), \ldots, b_\tau(z), \ldots, b_m(z))$ be the binary representation of the symbol z, where τ denotes the bit-plane representation of z and $b_\tau(s)$ the bit corresponding to this representation. The *multilevel* decoding is based on belief propagation at each stage τ, and the bit-by-bit posterior marginal log-likelihood ratio is initialized with

$$\log\left(\frac{\Pr[b_\tau(z) = 0, b_{\tau+1}(z), \ldots, b_m(z) | b_1(z), \ldots, b_{\tau-1}(z), y]}{\Pr[b_\tau(z) = 1, b_{\tau+1}(z), \ldots, b_m(z) | b_1(z), \ldots, b_{\tau-1}(z), y]}\right)$$

where the conditioning is with respect to already decoded bit-planes and the received noisy symbol.

All experimental results are obtained using the output degree distribution $\Omega(x) = 0.007690 + 0.493570x + 0.166220x^2 + 0.072646x^3 + 0.082558x^4 + 0.056058x^7 + 0.037229x^8 + 0.055590x^{18} + 0.025023x^{64} + 0.003135x^{65}$.

2.3.1 Symmetric Channels

We consider the transmission over the $q-\mathrm{SC}(p)$. The $q-\mathrm{SC}(p)$ with parameter p, takes a q-ary symbol as its input and outputs either the unchanged input symbol, with probability $1-p$, or any of the other $q-1$ symbols, with probability $\frac{p}{q-1}$. The capacity of the $q-\mathrm{SC}(p)$ with crossover probability p is

$$\mathrm{Cap}(q-SC(p)) = \log_2(q) - h(p) - p\log_2(q-1)$$

bits per channel use, where $h(p)$ denotes the binary entropy function. The number of output symbols is related to the overhead ε by

$$N = (1+\varepsilon)\frac{mk}{\mathrm{Cap}(q-\mathrm{SC}(p))}$$

where $q = 2^m$.

Fig. 2.6 compares the SER of our schemes with q-ary HSP_{60} and MSP_{100} decoding over the field $\mathrm{GF}(2^4)$, for fixed overhead and varying $q-\mathrm{SC}(p)$ parameters. Our algorithm was tested by choosing for C_Φ the $[15,4]$-Hadamard code, the $[7,4]$-Hamming code and the $[4,4]$-code. The values of k and N are chosen to be 6000 and 15000, respectively.

Fig. 2.7 compares the SER of our schemes with q-ary HSP_{60} and MSP_{100} decoding over the field $\mathrm{GF}(2^4)$, for varying overheads and a fixed $q-\mathrm{SC}(p)$ parameter. Our algorithm was tested by choosing for C_Φ the $[15,4]$-Hadamard code, the $[7,4]$-Hamming code and the $[4,4]$-code. The values of k and p are chosen to be 6000 and 0.89, respectively.

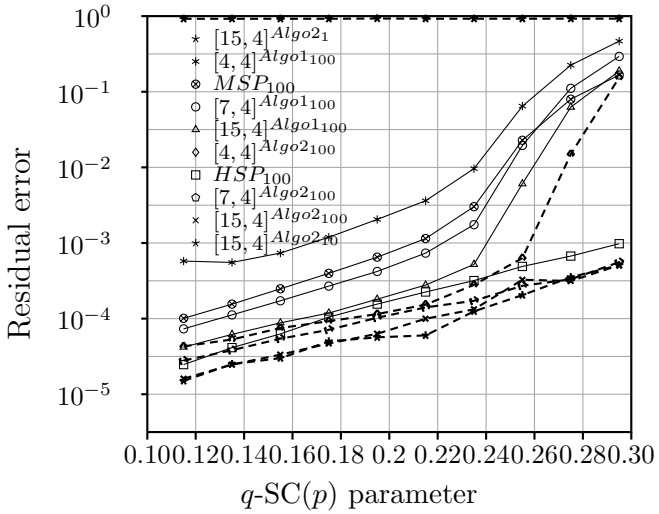


Figure 2.6: Comparison between our algorithms, HSP_{60} and MSP_{100} decoding over $\mathrm{GF}(2^4)$ for varying $q-\mathrm{SC}(p)$ parameters.

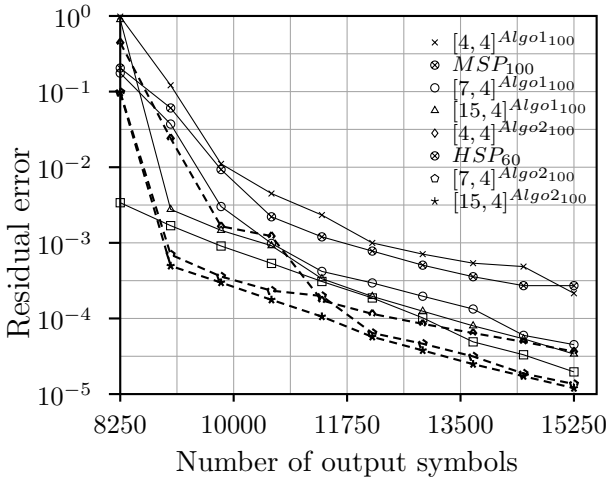


Figure 2.7: Comparison between our algorithms, HSP_{60} and MSP_{100} decoding over $\mathrm{GF}(2^4)$ for varying overheads and fixed $q-\mathrm{SC}(p)$ parameter.

Fig. 2.8 compares the SER of our schemes with q-ary HSP_{60} and MSP_{100} decoding over the field $\mathrm{GF}(2^8)$, for fixed overhead and varying $q-\mathrm{SC}(p)$ parameters. Our algorithm was tested by choosing for C_Φ the $[255, 8]$-Hadamard code, a $[29, 8]$-code and a $[17, 8]$-code. The $[17, 8]$-code is obtained by shortening a cyclic $[18, 9, 6]$-code. The 8×17 generator matrix $G(C_\Phi)$ for this $[17, 8]$-code C_Φ is given by

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

The $[29, 8]$-code is obtained by shortening an extended $[32, 11, 12]$-BCH code on three positions.The 8×29 generator matrix $G(C_\Phi)$ for this $[29, 8]$-code C_Φ is given by

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

Both these codes are optimal in the sense that they have the largest minimum distance given their blocklength and dimension. The values of k and N are chosen to be 4000 and 10000, respectively.

Fig. 2.9 compares the SER of our schemes with q-ary HSP_{60} and MSP_{100} decoding over the field GF(2^8), for varying overheads and a fixed $q-$SC(p) parameter. Our algorithm was tested by choosing for C_Φ the $[255, 8]$-Hadamard code, a $[29, 8]$-code and a $[17, 8]$-code, defined above.

As can be seen from the figures, there is a monotonic improvement as we increase the number of linear forms related to C_Φ, for a given order of the field. Also for finite block-length simulations, the second variant of our decoding algorithm (Algo. 9) performs better than the first variant. Moreover, our algorithm has a decoding performance close to that of SP as we increase the order of the field; the algorithms become closer in terms of decoding performance while maintaining a significant gap in terms of complexity. For example, for channels over the field GF(2^8), the $[255, 8]$-Hadamard code and the chosen $[29, 8]$-code for the proposed algorithm have symbol error rates that approach remarkably that of the q-ary SP with much lower complexity than the q-ary SP algorithm.

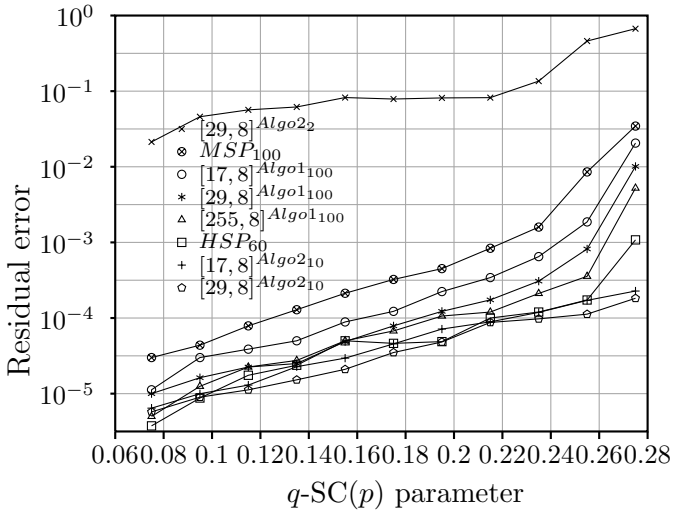


Figure 2.8: Comparison between our algorithms, HSP_{60} and MSP_{100} decoding over GF(2^8) for varying $q-$SC(p) parameters.

2.3.2 Non-Symmetric Channels

In this section, we consider a family of q-ary non-symmetric channels derived from PAM signal constellations transmitted on the AWGNC (Additive White Gaussian Noise Channel). We consider power and bandwidth efficient communication over the AWGNC which is defined by

$$Y = Z + T,$$

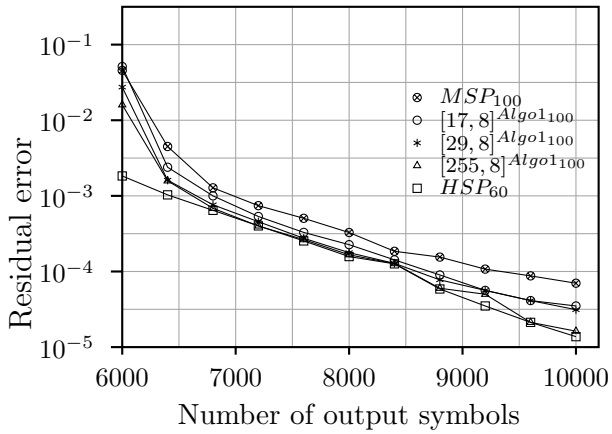


Figure 2.9: Comparison between our algorithms, HSP_{60} and MSP_{100} decoding over $\mathrm{GF}(2^4)$ for varying overheads and fixed $q-\mathrm{SC}(p)$ parameter.

where the channel input Z is disturbed by a random variable T which has a zero-mean Gaussian distribution with variance σ^2.

The energy expanded per channel use is equal to the mathematical expectation of Z^2 : $E_s = E[Z^2]$.

The capacity of the AWGN channel is given by the formula

$$C_{AWGN} = \frac{1}{2}\log_2(1 + \mathrm{SNR})$$

bits per use, where SNR is the signal to noise ratio and is defined as

$$\mathrm{SNR} = \frac{E_s}{\sigma^2},$$

We transmit Z from the PAM constellation S given by

$$S = \{-2^m + 2i + 1 | i = 0, \cdots, 2^m - 1\}.$$

The probability of having sent z provided that y is received is given by

$$\Pr(Z = z \mid Y = y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-z)^2}{2\sigma^2}}.$$

Fig. 2.10compare the SER of our schemes with q-ary HSP_{60} decoding over the field $\mathrm{GF}(2^4)$, for fixed overhead and varying SNRs. Our algorithm was tested by choosing for C_Φ the $[15, 4]$-Hadamard code, the $[7, 4]$-Hamming code and the $[4, 4]$-code. The values of k and N are chosen to be 6000 and 15000, respectively.

Fig. 2.11 compare the SER of our schemes with q-ary HSP_{60} decoding over the field $\mathrm{GF}(2^4)$, for varying overheads and a fixed SNR. Our algorithm was

tested by choosing for C_Φ the $[15, 4]$-Hadamard code, the $[7, 4]$-Hamming code and the $[4, 4]$-code. The values of k and SNR are chosen to be 6000 and 19.3 (in dB), respectively.

We remark from the figures, that there is still a monotonic improvement as we increase the number of linears forms related to C_Φ, as observed for q-ary symmetric channels. However the q-ary SP performs better for non symmetric channels.

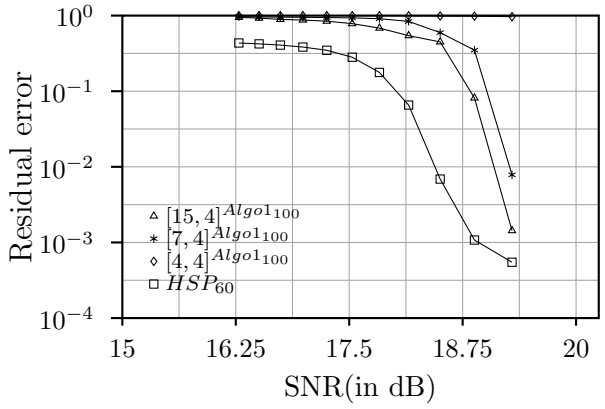


Figure 2.10: Comparison between our algorithms and HSP_{60} decoding over $\mathrm{GF}(2^4)$ for fixed overhead and varying SNRs.

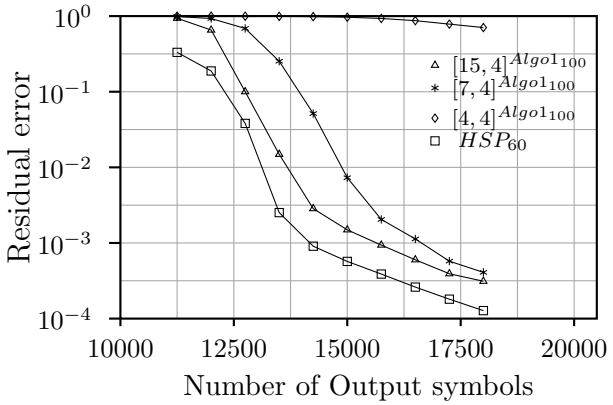


Figure 2.11: Comparison between our algorithms and HSP_{60} decoding over $\mathrm{GF}(2^4)$ for varying overheads and fixed SNR.

2.3.3 Conclusions

We provided a class of algorithms for decoding binary LT codes on q-ary channels. In contrast to the well-known q-ary SP algorithm, the new algorithm

provides a tradeoff between the complexity and the decoding capability of the algorithm. Several simulation results have been provided to show the performance of the new algorithms as compared to the q-ary SP algorithm.

It is important to note that our algorithms cannot be used when the underlying code is not binary (i.e., if it is not the tensor product of a binary code with GF(q)). On the other hand, one cannot expect binary codes to perform as well as q-ary ones, so that the question arises to what extent the algorithms of this paper are interesting for practical code design. One answer to this question is that our algorithms are expected to perform well on non-standard channels, for example channels that arise in practical loss-less and lossy coding algorithms.

Our algorithms can be extended in several directions. It has surely not escaped the attention of the reader that the proposed algorithms is applicable to any binary linear code, not just LT codes. In particular, if we have a binary code with an efficient binary decoding algorithm, we can use the same code and the methods proposed here to decode the code when used with q-ary symbols rather than binary ones. These schemes can be used for any binary code with an efficient soft-decision decoder; thus, for example, one could use convolutional codes with trellis decoding, LDPC and Turbo codes with quantized SP, Raptor codes with quantized SP.

LT Codes on Piecewise Stationary Channels

3

In Chapter 1, we gave descriptions of SP decoding algorithms that are used in the decoding process of LT codes for stationary binary memoryless symmetric channels with assumed known channel statistics. In this chapter, we provide algorithms for joint LT estimation-decoding over Piecewise Stationary Memoryless Channels (PSMC's) with a bounded number of abrupt changes in channel statistics. In particular, as a class of PSMC's, binary symmetric channels are considered with a crossover probability that changes a bounded number of times with no repetition in the statistics. We also provide joint LT estimation-decoding over Markov-modulated channels. Simulation results are given which illustrate the benefits of using our algorithms, both in terms of probability of error and in terms of redundancy.

3.1 Piecewise Stationary Memoryless Channels

3.1.1 Introduction

Many communication channels such as the well-known Gilbert-Elliott channel [16, 38], or more general finite-state Markov channel [19] models explicitly allow for sudden changes in the channel state during a transmission period. An abruptly changing channel model may be used to describe any system that experiences sudden, and persistent, bursts of interference or degradation, such as a mobile wireless channel, an optical CDMA channel, or a military communication channel in the presence of jamming. In many cases, the Markov channel model is useful for abruptly changing channels, but its use implies that the abrupt changes in the channel state have a statistical structure that is well-modeled by a hidden Markov chain. In many cases of interest, this assumption is not correct; for instance, there is no good statistical model that

reflects the possibility that a mobile terminal enters a tunnel or passes behind a building. A general model-free approach to abruptly changing channels, known as piecewise-symmetric memoryless channels (PSMCs), was given for binary-output channels in [52], and for Gaussian channels in [51]. In these works, a method from source coding, intended for abruptly changing sources [44], was adapted to give a channel estimation algorithm, which was incorporated into iterative decoding using Turbo codes. Although it was shown that large gains could be obtained by estimating the channel in this manner, the use of Turbo codes ([52],[51]) did not make use of the full power of the algorithms from [44], since their rates were not adaptive.

The main contributions of this chapter are threefold. First, we implement a version of this estimation algorithm for LT codes [17]for piecewise-symmetric memoryless channels. In particular, the variable rates of the LT code require us to make novel modifications to the algorithm in [44] to account for them, which were not required in the fixed-block-length scenario using Turbo codes in [52, 51]. Next, we study a quantized version of the SP for piecewise-symmetric memoryless channels. This algorithm was introduced by Gallager[18] for regular ldpc codes, and generalized to irregular ldpc codes[33] and LT codes[37] for symmetric channels. We derive density evolution equations to compute the probability of error. Finally, we generalized joint LT estimation-decoding algorithms for Markov-modulated channels.

3.1.2 Models and Definitions

First, a brief remark about notation. Realizations of a random variable X will be denoted by x. If a random process contains a sequence $X_0, X_1, \ldots$ of random variables, a vector of these variables will be denoted by $X_i^j = [X_i, X_{i+1}, \ldots, X_j]$ for $j > i$, with realization $x_i^j = [x_i, x_{i+1}, \ldots, x_j]$.

Consider a channel with random input $X_t \in \{+1, -1\}$, and random output $Y_t \in \mathcal{Y}$, where $t = 0, 1, 2 \ldots$ represents a time index. At each time t, the conditional probability function of Y_t given X_t is a function of some parameter ν_u denoted by

$$\Pr[Y_t = y_t \mid X_t = x_t; \nu_u] \triangleq \Pr[y_t \mid x_t; \nu_u],$$

where $u = 1, 2, \ldots, |\mathcal{T}|$ and $\mathcal{T}$ is defined below. We call $\Pr[y_t \mid x_t; \nu_u]$ the *family* corresponding to the PSMC. Throughout this chapter $\Pr[y_t \mid x_t; \nu_u]$ belongs to the binary symmetric channel family, with crossover probability ν_uf.

A PSMC is a channel where X and Y have the following properties:

- There exists a set $\mathcal{T}$ of transition points,

$$\mathcal{T} = [\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}|}],$$

  such that $\nu_u \neq \nu_{u+1}$ if and only if $u \in \mathcal{T}$. That is, ν changes at transition points, and nowhere else.

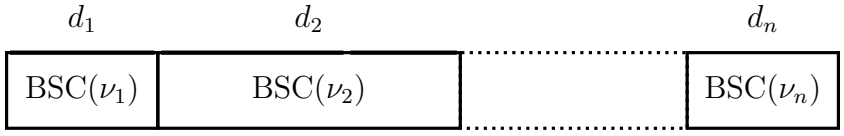


Figure 3.1: Graphical representation of a PSM-BSC

- If ν_i^j is known, then

$$\Pr[y_i^j \mid x_i^j; \nu_i^j] = \prod_{t=i}^{j} \prod_{u \in \mathcal{T}} \Pr[y_t \mid x_t; \nu_u]. \tag{3.1}$$

The first property captures the piecewise stationary nature of the channel, since the channel parameter remains constant until a time in $\mathcal{T}$ is reached. The second property captures the memoryless nature of the channel, since (3.1) represents a memoryless channel, so long as the channel parameters are all known.

Example 6. *Let* $\Pr[y_t|x_t; \nu_u]$ *represent a BSC family, such that* $\mathcal{Y} \in \{+1, -1\}$*, and* ν_t *represent the crossover probability. That is,*

$$\Pr[y_t \mid x_t; \nu_u] = \begin{cases} 1 - \nu_u, & y_t = x_t; \\ \nu_u, & y_t \neq x_t. \end{cases}$$

Suppose $\mathcal{T} = \{\tau_1\}$*, i.e., there is exactly one transition point at time* τ_1*. Letting* ν_1 *and* ν_2 *represent the crossover probability before and after* τ_1*, respectively, and for* $N > \tau_1$*, we may write*

$$\Pr[y_1^N \mid x_1^N; \nu_1^N] = \prod_{t=1}^{\tau_1} \Pr[y_t \mid x_t; \nu_1] \prod_{t=\tau_1+1}^{N} \Pr[y_t \mid x_t; \nu_2].$$

■

From the example, we see that we may parameterize a PSMC by $\mathcal{T}$, and a $(|\mathcal{T}| + 1)$-dimensional vector of channel behaviors ν. We will write R to represent such a vector, so that any PSMC from a given family is parameterized by $(\mathcal{T}, R)$. In Example 6, we have that $(\mathcal{T}, R) = (\{\tau_1\}, [\nu_1, \nu_2])$.

In this work, a PSM-BSC$(\mathcal{T}, R)$ (see Fig. 3.1) denotes a BSC channel divided into $n = |\mathcal{T}| + 1$ consecutive BSC channels, with crossover probabilities $\nu_1, \nu_2 \cdots \nu_n$ and respective fractional durations $d_1, d_2, \ldots, d_n$, where $\sum_{i=1}^{n} d_i = 1$. The number of abrupt transitions between BSC(ν_i) channels is assumed to be fixed and also the cross-over probability ν_i is constant on each BSC(ν_i).

We use the following notation.

- The LT-code with input parameters (k, Ω) has N output bits.

- β is the average degree of the output symbols.
- α is the average degree of an output node. $\alpha^{(j)}$ denotes the average degree of an input symbol connected to an ouput node related to BSC(ν_j).
- $I(x)^{(j)}$ is the node degree distribution of input nodes connected to ouput nodes related to BSC(ν_j).
- $\iota(x)^{(j)}$ is the edge degree distribution of input nodes connected to ouput nodes related to BSC(ν_j),

where $j \in \mathcal{I}_1^n$ refers to the channel number

Proposition 1. *Let an LT code with parameters (k, Ω) be given. Suppose that we have collected N output bits produced by this LT code sent over a PSM-BSC parameterized by $(\mathcal{T}, R) = ([\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}|}], [\nu_1, \nu_2, \ldots, \nu_{|\mathcal{T}+1|}])$. Then*

- $\alpha^{(j)} = d_j \alpha$.
- $I(x)^{(j)}$*and* $\iota(x)^{(j)}$ *can be approximated by Poisson distribution generating functions of mean* $\alpha^{(j)}$.

Proof: The proof is organized in two parts.

- By looking only at input nodes connected to output nodes related to BSC(ν_j), it's immediate that $\alpha^{(j)} = d_j \alpha$.
- For every generated output symbol sent over a BSC(ν_j), fix an input symbol. The probability that this input symbol is a neighbor of the output symbol which have been sent over a BSC(ν_j) is $p_j = \frac{(\beta d_j)}{k} = \frac{\alpha^{(j)}}{N}$. By looking the subgraph induced by the connection between input nodes connected to output nodes related to BSC(ν_j) and those output nodes, the remainder of the proof follows from manipulations similar to those of [17](Proposition 1).

■

A row vector of k bits is encoded using a specific LT code $x = (x_1, \cdots, x_k)$ into the row vector $z = (z_1, \cdots, z_N)$. We denote by $y = (y_1, \cdots, y_N)$ the received row vector and by $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ an estimate of x obtained after the joint estimation-decoding. The transmission is done bit-by-bit and the value of N does not need to be known in advance. As shown in Fig. 3.2, the vector z is then transmitted over a particular PSM-BSC($\mathcal{T}, R$), whose parameters take value ν_0 from τ_1 to $\tau_2 - 1$, ν_1 from τ_2 to $\tau_3 - 1$, and so on. Finally, from time τ_{n-1} to $\tau_n \triangleq N + 1$, the PSM-BSC parameter is ν_n. The vectors $(y_{\tau_1}, \cdots, y_{\tau_2 - 1}), (y_{\tau_2}, \cdots, y_{\tau_3 - 1}), \cdots, (y_{\tau_{n-1}}, \cdots, y_N)$ will be referred as stationary segments, and correspondingly, $\nu_1, \nu_2, \cdots, \nu_n$ will be called segmental parameters.

In what follows, we describe the modifications to LT decoding algorithms

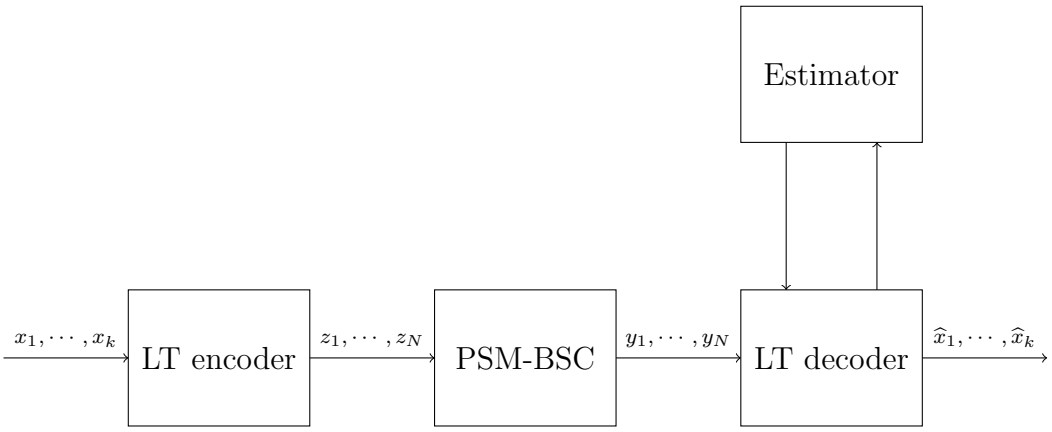


Figure 3.2: LT joint estimation-decoding system.

useful to use these codes on PSMCs with unknown parameters. First, we describe the estimation of channel statistics, assuming the transition points are known. Finally, we give several methods for estimating the location of the transition points assuming the transitions points are unknown. All of these algorithms are described in terms of jointly estimating the channel and decoding the LT code.

3.1.3 Estimation of Channel Statistics

During the decoding, we estimate the crossover probability ν in each PSM-BSC segment (each segment being a BSC channel), assuming that the transition points between these segments are known: $\widehat{\nu}$ is the crossover probability estimate of ν. The discussion about the location of transition points is left for the next section.

For $i \in \mathcal{I}_1^N$, if q_i denotes the LLR expressing a soft belief in the current value of $z_i \in \{0, 1\}$, and $\Pr[Z_i = z_i]$, an a-priori probability given by this extrinsic information, q_i is associated to $\Pr[Z_i = z_i]$ as

$$q_i \triangleq \log \frac{\Pr[Z_i = 0]}{\Pr[Z_i = 1]}.$$

The posteriori probability for each z_i, by assuming that the crossover probability ν is equal to its estimate $\widehat{\nu}$, is given by the formula

$$\Pr[z_i \mid y_i; \widehat{\nu}] \;=\; \frac{\Pr[y_i \mid z_i; \widehat{\nu}]\Pr[Z_i = z_i]}{\sum_{z_i \in \{0,1\}} \Pr[y_i \mid z_i; \widehat{\nu}]\Pr[Z_i = z_i]}.$$

The LLR of a received symbol y_i is defined as

$$Z^{(\widehat{\nu})} \triangleq \log \frac{\Pr[Z_i = 0 \mid y_i; \widehat{\nu}]}{\Pr[Z_i = 1 \mid y_i; \widehat{\nu}]}.$$

We describe two methods for estimating the statistics in a PSM-BSC segment. The first is based on the expectation-maximization ([13]) algorithm that has access to the complete segment for estimating the change point. The other is more restrictive and assumes only sequential access to the data for estimating the crossover statistics. We make the following assumptions :

- The set of transition points $\mathcal{T}$ is known.
- The capacity Cap(PSM-BSC($\mathcal{T}, R$)) of the given PSM-BSC($\mathcal{T}, R$) is known.
- The sequence of extrinsic messages q is independent of R and y.

3.1.4 Sequential Channel Estimation Algorithm

The first method of estimating the statistics in a PSM-BSC segment is an adaptation of an algorithm that was recently proposed in [46] for sequential probability estimation. The algorithm generalizes the well-known Krichesvky-Trofimov(KT) [28] sequential estimator in the binary case for parameters that are confined within a smaller interval than the standard $[0, 1]$ interval, i.e. crossover can only take values smaller than $1/2$.

In the context of compressing binary sequences, if $n > 0$, $t \in \mathcal{I}_1^{n-1}$, $s^t = (s_1, \cdots, s_t)$ a sequence of independent and identical distributed bits, $0 < \alpha < \beta < 0.5$, $n^t(b)$ is the occurrence count of bit b in the sequence s^t, then the probability assigned to a binary sequence $s = (s_1, \cdots, s_n)$ given by the generalized KT estimator can be updated as

$$\begin{aligned} \Pr[s^{t+1}] &= \Pr[s^t]\frac{n^t(s_{t+1}) + 0.5}{(t+1)} + \\ &\quad -\frac{\alpha^{n^t(1)+0.5}(1-\alpha)^{n^t(0)+0.5}}{C \cdot (t+1)} - \\ &\quad -\frac{\beta^{n^t(1)+0.5}(1-\beta)^{n^t(0)+0.5}}{C \cdot (t+1)} \end{aligned} \tag{3.2}$$

where $\Pr[y^0 = 1]$ is the initial step and $C = 2(\sin^{-1}\sqrt{\beta} - \sin^{-1}\sqrt{\alpha})$.

In the case $\alpha = 0, \beta = 1$, $C = \pi$, then (3.2) reduces to the binary form of the standard KT ([28]) estimator given as ,

$$\Pr[s^{t+1}] = \Pr[s^t]\frac{n^t(s_{t+1}) + 0.5}{(t+1)}. \tag{3.3}$$

In our context (channel estimation), we would like to use the soft count on each assumed memoryless BSC(ν_j), or more generally on a segment j of channel, to estimate the probability of the next bit, where $j \in \mathcal{I}_1^n$.

If we use the standard KT(3.3) approach, we can use the soft count to estimate the probability of the next bit. This is because the KT recursion

gives an explicit expression of the probability of the next bit, that depends only on its occurrences thus far. So, we can count the bits (using their soft values) in each time unit, adding 1/2 to each of the counts, and estimating the probability like that. This is a soft add-$\frac{1}{2}$ estimator for the probability of the next bit. The difficulty, however, is if we are on a limited interval which is not $[0, 1]$. Then, this will give an initial estimate for each probability of $\frac{1}{2}$. Thus, we need an estimator of the form presented in (3.2). The problem with the estimator (3.2) is the recursion format of the equation. The probability of the next bit must be computed as the ratio of the probability of adding one to each count compared to the previous counts. Since we do soft counts, we do not add 1 in each step because each bit occurs a fraction of 1 times at a given time unit, according to its soft estimate. This is what ruins the possibility to use the equation (3.3) in a soft count scenario, because there is not really a recursion if the 1 bit added is spread between a part 0 and a part 1.

One solution is as follows. Since we can do something like standard KT in the soft decision setting, we can switch to it after the KT estimator is relatively reliable. So, we can code, say, the 100 or even 10 first bits using the estimator (3.2), and then switch to the soft add-$\frac{1}{2}$ estimator for the remainder of the hypothesized segment. This can be done for every state representing a new segment. Now, during the first bits, we either use hard decisions or do the following. The Generalized KT estimator is a function of the number of occurrences. It does not matter in which order things occur. So, at each time point when we come up with a soft count, we can round its value to the closest integer counts. Then, we do the complete recursion with the rounded values of the counts to get the estimate of the next probability.

3.1.5 The Expectation Maximization Method

The second crossover estimation technique in a PSM-BSC segment is based on the expectation-maximization ([13]) method. The expectation-maximization (EM) is an iterative optimization method to compute the maximum likelihood estimates in the presence of missing or hidden data in probabilistic models. This method is used for learning for variables whose value is never directly observed.

In what follows, we apply the EM ([13]) to estimate the crossover probability ν_j on each channel BSC(ν_j) given the received sequence y and the sequence of extrinsic messages (obtained from the SP decoding) $q = [q_1, \cdots, q_N]$. Using the terminology of [13] , we designate y the *incomplete* data, $[x, y]$ the *complete* data, and $R = [\nu_1, \nu_2, \ldots, \nu_n]$ the set of parameters to estimate.

The EM method described here proceeds in four main steps: the initialization (Init-step), the Sum-Product (SP-step), the expectation step (E-step) and the maximization step (M-step). In what follows, $\nu_{(\ell,j)}$ is the crossover probability of a BSC(ν_j) at step number $\ell \geq 0$, where $j \in \mathbb{J}_1^n$, $\widehat{\nu}_{(\ell,j)}$ is estimate.

Algorithm 10. *EM method*

Input: Row vector of noisy output bits $y = (y_1, \cdots, y_N)$ *and the corresponding graph for an LT code with parameters* (k, Ω).
Output: Row vector of crossover probability estimates $\widehat{R} = [\widehat{\nu_1}, \widehat{\nu_2}, \ldots, \widehat{\nu_n}]$.

- Init-step: *First, at step* $\ell = 0$, *arbitrarily choose an estimate* $\widehat{\nu}_{(0,j)}$ *of* ν_j. *It does not matter how this estimate is chosen; it could be a wild guess. In this work, we obtain* $\widehat{\nu}_{(0,j)}$ *by making the wrong assumption that the given PSM-BSC*$(\mathcal{T}, R)$ *is a stationary BSC*$(\widehat{\nu}_{(0,j)})$,.

- SP-step: *Perform several SP decoding iterations. Messages are passed at each round* $d \geq 0$ *as*

$$M_{inb^i \to outb^j}^{(d,\nu_{(\ell,a)})} = \sum_{b=1}^{n} \sum_{p=1}^{i-1} M_{inb^i \leftarrow outb^{j'_p}}^{(d,\nu_{(\ell,b)})},$$

$$M_{inb^i \leftarrow outb^j}^{(d+1,\nu_{(\ell,a)})} = 2\text{arctanh}\left(\tanh\left(\frac{Z^{(\nu_{(\ell,a)})}}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M_{inb^{i'_l} \to outb^j}^{(d,\nu_{(\ell,a)})}}{2}\right)\right),$$

where $M_{inb^i \leftarrow outb^{j'_1}}^{(d,\nu_{(\ell,b)})}, \cdots, M_{inb^i \leftarrow outb^{j'_{i-1}}}^{(d,\nu_{(\ell,b)})}$ *are messages related to all output nodes* $outb^{j'_p}$, *related to BSC*$(\nu_{(\ell,b)})$, *adjacent to* inb^i *other than* $outb^j$, $p \in \mathcal{I}_1^{i-1}$; $M_{inb^{i'_l} \to outb^j}^{(d,\nu_{(\ell,a)})}, \cdots, M_{inb^{i'_{j-1}} \to outb^j}^{(d,\nu_{(\ell,a)})}$ *are messages related to all input nodes* $inb^{i'_l}$ *connected to output nodes related to BSC*$(\nu_{(\ell,a)})$, *adjacent to* $outb^j$ *other than* inb^i, $l \in \mathcal{I}_1^{j-1}$, $Z^{(\nu_{(\ell,a)})}$ *being the initial LLR related to BSC*$(\nu_{(\ell,a)})$, $a \in \mathcal{I}_1^n$, $b \in \mathcal{I}_1^n$, $j'_p \in \mathcal{I}_1^k$ *and* $i'_l \in \mathcal{I}_1^N$.

- E-step: *For each stationary segment* $(y_{\tau_a}, \cdots, y_{\tau_{a+1}-1})$, *calculate*

$$Q(\nu|\widehat{\nu}_{(\ell,a)}) \triangleq \mathrm{E}\big[\log \Pr[y_{\tau_a}^{\tau_{a+1}-1}, z_{\tau_a}^{\tau_{a+1}-1}; \nu]|y_{\tau_a}^{\tau_{a+1}-1}\big], \tag{3.4}$$

where $a \in \mathcal{I}_1^{n-1}$ *and the subscript* $\nu_{(\ell,a)}$ *indicates that the expectation is taken under the assumption that* $\nu = \widehat{\nu}_{(\ell,a)}$.
We can dispose of any terms of $Q(\nu|\widehat{\nu}_{(\ell,a)})$ *that are not functions of* ν, *so the expectation becomes*

$$Q(\nu|\widehat{\nu}_{(\ell,a)}) = \mathrm{E}\big[\log \Pr[z_{\tau_a}^{\tau_{a+1}-1}|y_{\tau_a}^{\tau_{a+1}-1}; \nu]|y_{\tau_a}^{\tau_{a+1}-1}\big]. \tag{3.5}$$

Now the expression (3.4) is written as

$$\begin{aligned}
Q(\nu|\widehat{\nu}_{(\ell,a)}) &= \sum_{z_{\tau_a}^{\tau_{a+1}-1}\in\{0,1\}^{\tau_{a+1}-\tau_a}} \Pr[z_{\tau_a}^{\tau_{a+1}-1}|y_{\tau_a}^{\tau_{a+1}-1};\widehat{\nu}_{(\ell,a)}]\log\Pr[z_{\tau_a}^{\tau_{a+1}-1}|y_{\tau_a}^{\tau_{a+1}-1};\nu] \\
&= \sum_{z_{\tau_a}^{\tau_{a+1}-1}\in\{0,1\}^{\tau_{a+1}-\tau_a}} \prod_{i=\tau_a}^{\tau_{a+1}-1}\Pr[z_i|y_i;\widehat{\nu}_{(\ell,a)}]\sum_{i=\tau_a}^{\tau_{a+1}-1}\log\Pr[z_i|y_i;\nu] \\
&= \sum_{i=\tau_a}^{\tau_{a+1}-1}\sum_{z_i\in\{0,1\}}\Pr[z_i|y_i;\nu_{(\ell,a)}]\log\Pr[z_i|y_i;\nu] \\
&= \sum_{i=\tau_a}^{\tau_{a+1}-1}(\log(1-\nu))\Pr[z_i=y_i|y_i;\widehat{\nu}_{(\ell,a)}] + \\
&\quad (\log\nu)(1-\Pr[z_i=y_i|y_i;\widehat{\nu}_{(\ell,a)}]) \\
&= (\log(1-\nu)\sum_{i=\tau_a}^{\tau_{a+1}-1}\Pr[z_i=y_i|y_i;\widehat{\nu}_l] + \\
&\quad (\log\nu)\sum_{i=\tau_a}^{\tau_{a+1}-1}(1-\Pr[z_i=y_i|y_i;\widehat{\nu}_{(\ell,a)}])
\end{aligned} \tag{3.6}$$

where the fourth line follows from the fact that z_i is either the same as y_i, or its opposite value.

- M-step: *For each stationary segment, find the value of $\nu_{\max}$ maximizing $Q(\nu|\widehat{\nu}_{(\ell,a)})$. This value is given as the average over all posteriori probabilities that each bit is decoded correctly, i.e.,*

$$\widehat{\nu}_{\max} = \frac{1}{(\tau_{a+1}-\tau_a)}\sum_{i=\tau_a}^{\tau_{a+1}-1} f(z_i\neq y_i \mid y_i;\widehat{\nu}_{(\ell,a)}).$$

  Finally, we set $\widehat{\nu}_{(\ell+1,a)} = \widehat{\nu}_{\max}$.

- *Let $\ell := \ell+1$, and go to the* E-step *until convergence.*

Now for each z_i, we can calculate $\Pr[z_i \mid y_i;\widehat{\nu}_{(\ell,a)}]$, where $i \in \mathcal{I}_{\tau_a}^{\tau_{a+1}-1}$.

In what follows, we present two segmentation strategies(appeared in [44]-[45]), one that partitions the data into equal length blocks, assuming changes between the blocks. The other employs a recursive algorithm to identify a point that is relatively close to the change. We assume the following:

- The set of transition points $\mathcal{T}$ is unknown.
- The capacity of the given PSM-BSC$(\mathcal{T},R)$, Cap(PSM-BSC$(\mathcal{T},R)$) is known.
- The sequence of extrinsic messages q is independent of R and y.

3.1.6 Decoding using the Block Partitioning segmentation

The Block Partitioning (BP) algorithm partitions the data into blocks of equal length and estimate the parameters within each block, under two assumptions:

- The parameter within the block is always the same (which is actually not true if a transition point occurs within the block, but we consider this acceptable to keep the algorithm simple);
- Parameters within different blocks are independent.

The length can be determined to minimize the tradeoff between two types of penalties. One is caused by lack of statistics for blocks of insufficient length. The other is caused by the worst case in which a change occurs in the midpoint of a block. The first requires longer blocks, while the second requires shorter ones. Roughly blocks of the order of $\sqrt{N}$ achieve optimal tradeoff (as in [45], $\sqrt{N}$ is a guideline, but we choose a fixed block length).

The LT estimation-decoding in conjunction with the BP works as follows:

- *Initialization phase*: We choose $\widehat{\nu}_{(0,j)}$ by making the assumption that the given PSM-BSC($\mathcal{T}, R$) is a stationary BSC($\widehat{\nu}_{(0,j)}$).
- *Sum-Product phase*: The decoder performs several sum-product iterations.Messages are passed at each round $d \geq 0$ as

$$M^{(d,\nu_{(\ell,a)})}_{inb^i \to \mathrm{outb}^j} = \sum_{b=1}^{n} \sum_{p=1}^{i-1} M^{(d,\nu_{(\ell,b)})}_{inb^i \leftarrow outb^{j'_p}},$$

$$M^{(d+1,\nu_{(\ell,a)})}_{inb^i \leftarrow outb^j} = 2\mathrm{arctanh}\left(\tanh\left(\frac{Z^{(\nu_{(\ell,a)})}}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M^{(d,\nu_{(\ell,a)})}_{inb^{i'_l} \to \mathrm{outb}^j}}{2}\right)\right),$$

  where $M^{(d,\nu_{(\ell,b)})}_{inb^i \leftarrow outb^{j'_1}}, \cdots, M^{(d,\nu_{(\ell,b)})}_{inb^i \leftarrow outb^{j'_{i-1}}}$ are messages related to all output nodes $outb^{j'_p}$, related to BSC($\nu_{(\ell,b)}$), adjacent to inb^i other than outb^j, $p \in \mathcal{I}_1^{i-1}$; $M^{(d,\nu_{(\ell,a)})}_{inb^{i'_1} \to outb^j}, \cdots, M^{(d,\nu_{(\ell,a)})}_{inb^{i'_{j-1}} \to outb^j}$ are messages related to all input nodes $inb^{i'_l}$ connected to output nodes related to BSC($\nu_{(\ell,a)}$), adjacent to outb^j other than inb^i, $l \in \mathcal{I}_1^{j-1}$, $Z^{(\nu_{(\ell,a)})}$ being the initial LLR related to BSC($\nu_{(\ell,a)}$), $a \in \mathcal{I}_1^n$, $b \in \mathcal{I}_1^n$, $j'_p \in \mathcal{I}_1^k$ and $i'_l \in \mathcal{I}_1^N$.
- *Transition Point Estimation phase*: Next, the decoder applies the *BP* on the noisy sequence y_1^N. The sequence y_1^N is partitioned into equal length segments.
- *PSM-BSC statistics estimation phase*: An estimate of crossover probabilities $\nu_1, \cdots, \nu_n$ is obtained by using the soft values of $\widehat{x}_1^N$ and the received noisy sequence y_1^N. This crossover estimation is done by using either the EM algorithm, or the sequential channel estimation algorithm discussed above, on each of the obtained PSM-BSC segments.

- Go to the *Sum-Product phase* phase. This process stops when successful decoding is achieved, or when the maximum number of iterations has expired.

3.1.7 Decoding using a Recursive Decision segmentation

The recursive decision (RD) algorithm attempts to estimate up to a bounded number of changes within the error sequence hypothesized by the current estimates of the vector denoted $\widehat{X}_1^N$. The parameter S, provided to the algorithm, determines the total number of distinct transition estimates to be obtained. The components of the error sequence are the current a-posteriori probabilities that $x_i \neq y_i$. The algorithm, taken from [44], functions iteratively at levels, starting with level $\ell > 0$. The hypothesized (soft) error sequence is partitioned into blocks, referred as β_ℓ, of equal length $\kappa_\ell > 0$, for some top level ℓ. Then for each block β_ℓ , the statistical distance, called metric, is measured between the data in the two blocks surrounding the block. These results are used to divide the blocks β_ℓ with the highest metrics into sub-blocks $\beta_{\ell-1}$, of length $\kappa_{\ell-1}$ (generally an integer divisor of κ_ℓ), for which the metrics are calculated again, and the highest metric blocks subdivided. The algorithm terminates at level 0, where the transition point is estimated to occur in the middle of the block with the highest metric.

The metric for each block, at any level, is calculated as follows. Let $\Pr[\beta^1]$ and $\Pr[\beta^2]$ be the average bit error probabilities for blocks β^1 and β^2, respectively, found from the bit error probability estimation algorithm. The empirical entropy of the concatenation of blocks β^1 and β^2 is given by

$$H(\beta^1,\beta^2) = \frac{1}{2}\big((\Pr[\beta^1]+\Pr[\beta^2])\log\frac{1}{2}(\Pr[\beta^1]+\Pr[\beta^2]) + (2-\Pr[\beta^1]-\Pr[\beta^2])\log\frac{1}{2}(2-\Pr[\beta^1]-\Pr[\beta^2])\big).$$

The empirical entropy of the block β is given simply by $H(\beta,\beta)$. The metric $M(\beta)$ of a block β , is obtained from the empirical soft values of both neighboring blocks $\beta-1$ and $\beta+1$ as

$$M(\beta) \triangleq H(\beta-1,\beta+1) - \frac{1}{2}H(\beta-1,\beta-1) - \frac{1}{2}H(\beta+1,\beta+1).$$

If some block β has a large metric, it is likely that a change occurred within the block or its neighborhood. Theoretically, for the universal lossless coding point of view, Shamir and Costello [44] showed that if $\kappa_1 = O(\log(N))$ and $\kappa_0 = O(\log\log(N))$, two levels of the algorithm are sufficient, however, in practical applications, the algorithm can be performed with more levels, where $N > 0$ is the length of the hypothesized sequenceto to code. Figure 3.3 (taken from [43]) illustrates a three level segmentation mechanism for $\ell = 2$. For level $v \in \{0,1,2\}$, the level-v near neighborhood of block β_v is defined as the

concatenation of the data that constitute blocks $\beta_v - 1$, β_v and $\beta_v + 1$. An error sequence of length N, on which a level of the algorithm is performed, is represented by a bold horizontal line, which is partitioned into the respective blocks by vertical lines. Thick vertical lines in level-0 represent level-1 partitioning points and thick vertical lines in level-1 represent level-2 partitioning. On the top level (level-2), blocks β_2^1 and β_2^2 have the largest metrics. Level-1 is performed on each of the near neighborhoods of the two blocks. In Figure 3.3, level-1 is illustrated only for the level-2 near neighborhood of β_2^2. At level-1, β_1^3 and β_1^4 are the highest metrics blocks in the near neighborhood of level-2. Block β_1^4 has the largest metric in the near neighborhood of level-2 and block β_0^4 has the largest level-1 metric in the near neighborhood of level-1 block β_1^4. The transition point $\widehat{t_4}$ is thus estimated as the middle of the block β_0^4.

Estimation-Decoding in conjunction with the RD method follows the same steps as with the *BP* algorithm, modifying only the *Transition Point Estimation* step.

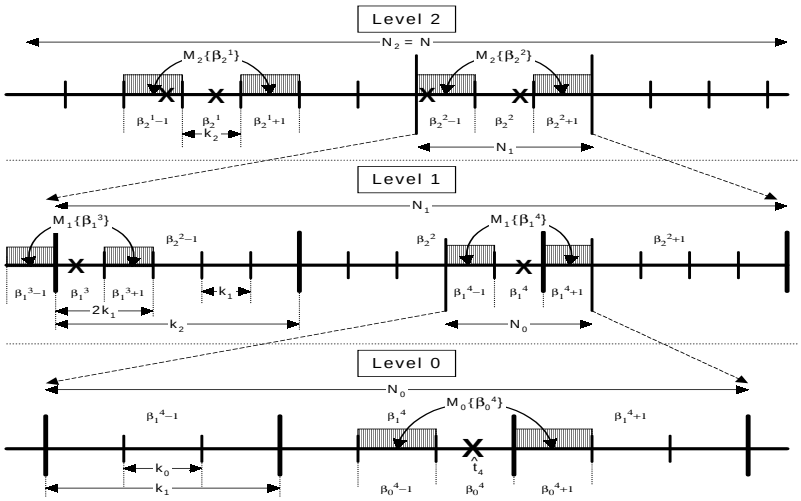


Figure 3.3: Recursive block partitioning for a three-level decision algorithm.

Example 7. *We illustrate the SP decoding calculations for a PSMC-BSC*($\{\tau_1\}$, $[\nu_1, \nu_2]$) *in Fig. 3.4, Fig. 3.5 and Fig. 3.6.* ■

3.1.8 Performance Comparison

In this section, we compare the performance of the proposed schemes with perfect knowledge of the PSM-BSC in terms of bit error rate (BER) and in terms of redundancy. All results were obtained using the output degree distribution Ω given by its generating function $\Omega(x) = 0.1629 + 0.3530x + 0.0941x^2 + 0.0455x^3 + 0.0942x^4 + 0.097x^5 + 0.0154x^{10} + 0.0875x^{11} + 0.0004x^{86} + 0.05x^{87}$. The length of the LT code must be at least N_{min} , where $N_{min} := \frac{k}{\text{Cap(PSM-BSC)}}$ is the capacity required length, and any additional bits are called redundant

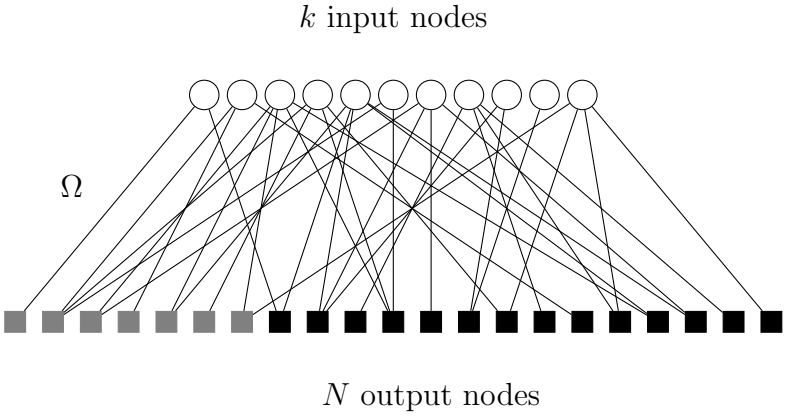


Figure 3.4: Decoding graph of an LT code for a PSMC-BSC($\{\tau_1\}, [\nu_1, \nu_2]$)

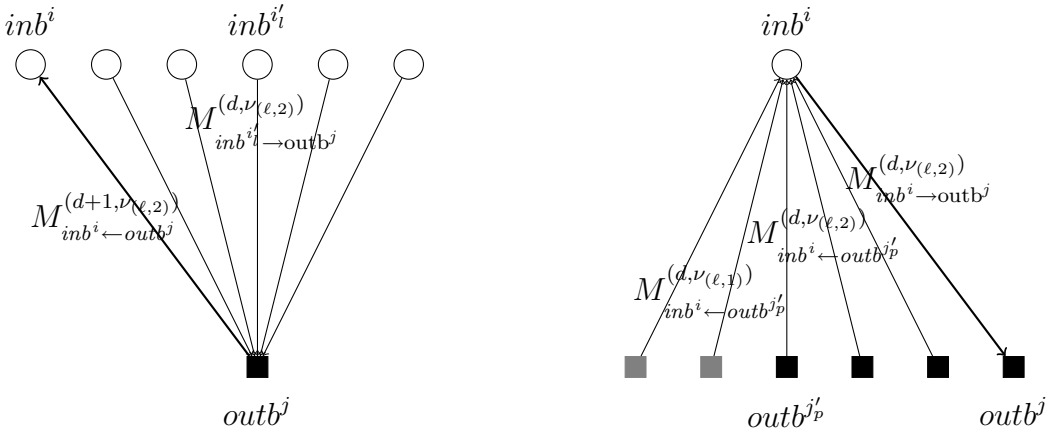


Figure 3.5: Message flow related to BSC($\{\tau_1\}$)

bits. A PSM-BSC is constructed in such a way that nd_i bits are sent through a BSC with crossover probability ν_i, and the r redundant bits are all sent through the channel with crossover probability $\nu_{|\mathcal{T}|+1}$. The length N_{min} of the considered LT code is thus obtained by $N_{min} = r + (\sum_{i=1}^{|\mathcal{T}|+1} N_{min} d_i) = r + N_{min}$. That is, the redundant bits are all sent in the nth channel, and no transitions are allowed while the redundant bits are being transmitted. The RDA algorithm is performed with 3 levels and parameterized by $\mathcal{L}$, where $\mathcal{L}^{\mathcal{RD}} = (\mathcal{L}_2, \mathcal{L}_1, \mathcal{L}_0)$. The BP algorithm is parametrized by $\mathcal{L}^{\mathcal{BP}}$ which represents the blocks length.

BER results

We compare the information BER performance of the system illustrated in Fig. 3.7 where full PSM-BSC statistics are unknown to which PSM-BSC statistics are perfectly known by the decoder. The LT code parameters $(k, \Omega) =$

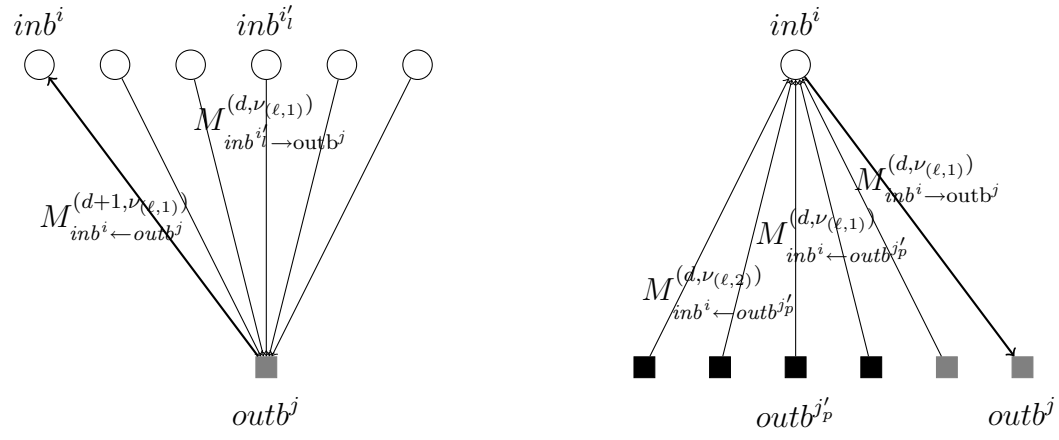


Figure 3.6: Message flow related to BSC($\{\tau_2\}$)

$(7000, \Omega^1)$ where $N = 20000$ (N is known in advance) and Ω^1 is defined previously. The PSM-BSC parameters are $(\mathcal{T}, R) = (\{5000, 8000\}, [0.25, 0.49, P3])$, where $0.01 \leq P3 \leq 0.1$. The RDA parameter is $\mathcal{L}^{\mathcal{RD}} = (500, 100, 20)$. The BP parameter is $\mathcal{L}^{\mathcal{BP}} = 100$.

Redundancy results

In Fig. 3.8, we compare the information redundancy performance of the decoder in which perfect PSM-BSC statistics are provided to the decoder with BP and RD algorithms. The LT code parameters $(k, \Omega) = (6000, \Omega)$, where Ω has been defined above. The PSM-BSC parameters are $(\mathcal{T}, R) = (\{6150, 10000\}, [0.305, 0.105, P3])$, where $0.415 \leq P3 \leq 0.46$. The RDA parameter is $\mathcal{L}^{\mathcal{RD}} = (500, 100, 20)$ and the BP one is $\mathcal{L}^{\mathcal{BP}} = 100$ as before.

We start with a fixed absolute length for segments 1 and 2. Then we add redundancy bits in segment 3 until decoding is complete. The stop criteria here is a BER of 0. Thus we compute the relative duration of each segment in the final block length. The capacity required length is then evaluated for the computed relative duration. The redundancy is the number of bits beyond this length.

Discussion

As can be seen in Fig. 3.7 and Fig. 3.8, the performance with estimation, using a recursive decision segmentation is very close to that with perfect knowledge of the channel statistics, and is better than using the block partitioning segmentation approach. These resuls are a demonstration of the power and potential approaches we proposed.

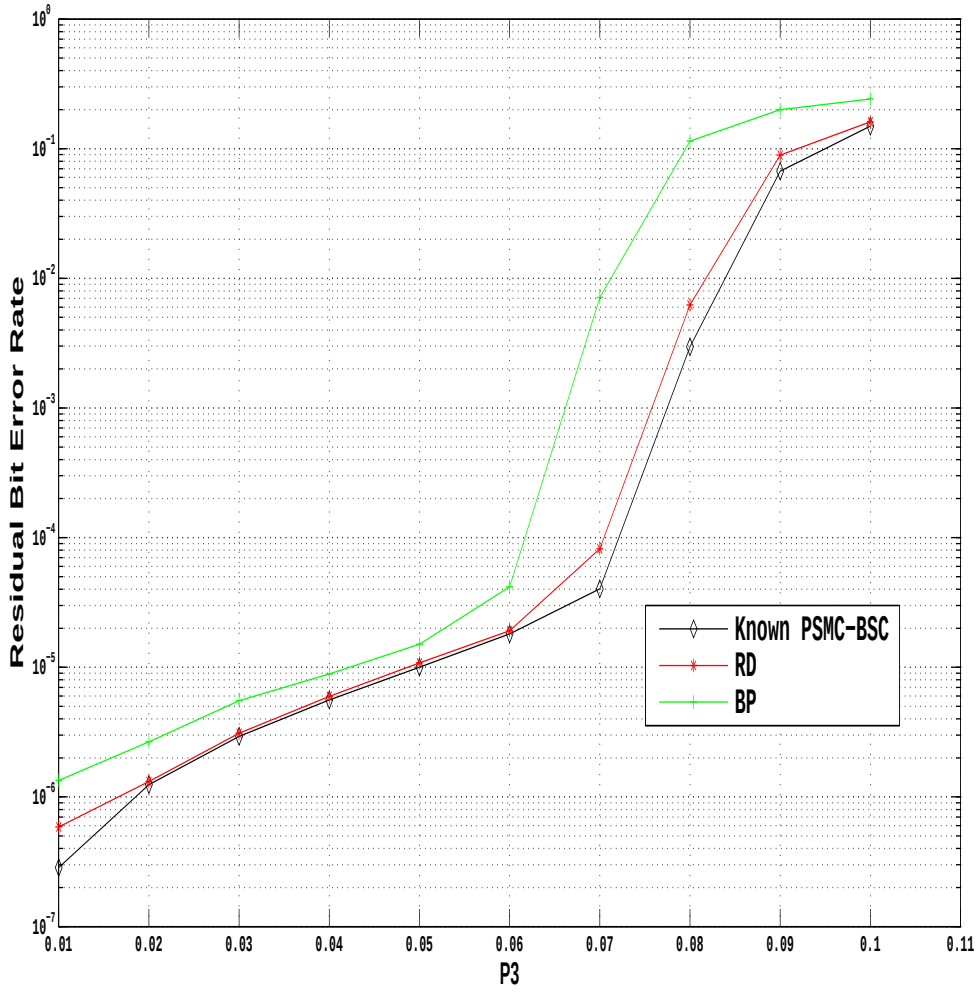


Figure 3.7: BER performance with varying bad channel.

3.1.9 Hard-Information Optimization

In this section, we investigate the performance of LT codes using Gallager's majority decoding ([18],[33],[37]) algorithms on PSM-BSC's channels. We will first describe this algorithm. Then we will obtain some recursive equations for the probability of error in terms of the degree distributions.

Gallager's Majority Decoding

We adapt the Gallager's majority decoding algorithm using LT-codes for PSM-BSC's channels. We first describe the different calculations that occur at each node, and subsequently we give the message passing schedules over the graph.

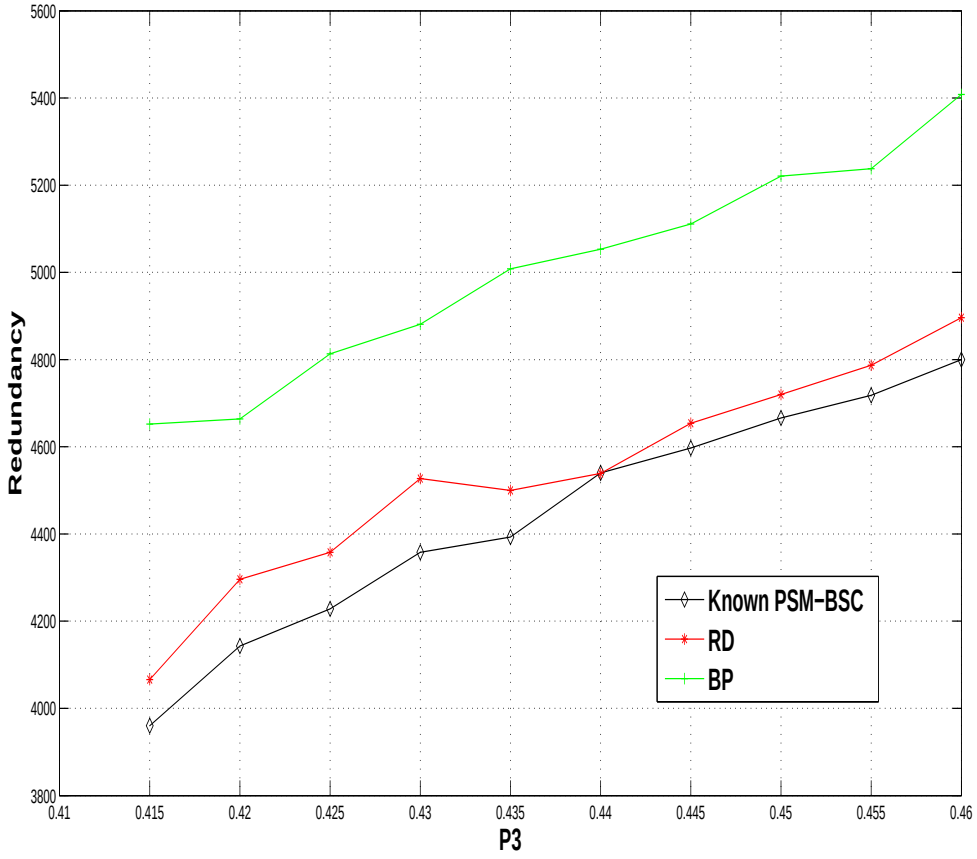


Figure 3.8: Redundancy performance with varying bad channel.

At the beginning of the decoding, each output node has value $+1$ or -1 according to the output of the PSM-BSC channel. The messages passed in this algorithm come from the alphabet $\{-1, 0, 1\}$, where 0 denotes erasure, and 1 and -1 correspond to the usual antipodal signalling, where 0 denotes an erasure [37].
We denote $C^{(\nu_a)} \in \{-1, 1\}$ as the channel observation for a given output node related to BSC(ν_a).

At the very first round of the decoding, where there is no message from the input nodes, output nodes with degree 1 send their values to their unique neighbours in the set of the input nodes. Other output nodes which can not estimate their neighbours value, send 0 to the input nodes.

Algorithm 11. *Gallager's Majority Decoding Algorithm*
Input: Row vector noisy output symbols $y = (y_1, \cdots, y_N)$ *and the corresponding graph for an LT code with parameters* (k, Ω).
Output: Row vector of estimates of input symbols $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. **Update rule for intput nodes.**
Set the components of the message $M^{(d,\nu_a)}_{inb^i \to outb^j}$ from input node i of degree d_i to output node j, related to BSC(ν_a) as follows :

$$M^{(d,\nu_a)}_{inb^i \to outb^j} = \text{sign}\left(\sum_{b=1}^{n} \sum_{p=1}^{i-1} M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_p}} \right)$$

where $M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_1}}, \cdots, M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_{i-1}}}$ are messages related to all output nodes $outb^{j'_p}$, related to BSC(ν_b), adjacent to inb^i other than $outb^j$, $p \in \mathfrak{I}_1^{i-1}$, $b \in \mathfrak{I}_1^n$, $j'_p \in \mathfrak{I}_1^k$.

2. **Update rule for output nodes.**
The message sent from an output node to an input node is equal to the channel output times the multiplication of the received message from input nodes in the last round of the algorithm. Set the components of the message $M^{(d,\nu_a)}_{inb^i \leftarrow outb^j}$ from output node j of degree d_j, related to BSC(ν_a), to input node i as follows:

$$M^{(d+1,\nu_a)}_{inb^i \leftarrow outb^j} = C^{(\nu_a)} . \prod_{l=1}^{j-1} M^{(d,\nu_a)}_{inb^{i'_l} \to outb^j}$$

where $M^{(d,\nu_a)}_{inb^{i'_1} \to outb^j}, \cdots, M^{(d,\nu_a)}_{inb^{i'_{j-1}} \to outb^j}$ are messages related to all input nodes $inb^{i'_l}$ connected to output nodes related to BSC(ν_a), adjacent to $outb^j$ other than inb^i, $l \in \mathfrak{I}_1^{j-1}$, $a \in \mathfrak{I}_1^n$ and $i'_l \in \mathfrak{I}_1^N$.

In practical settings, the decoder stops after a fixed number of SP decoding rounds and computes at each input node inb^i a maximum a posteriori estimate of the s^{th} transmitted information symbol x_s, where $s \in \mathfrak{I}_1^k$.

The estimation of the transmitted information symbol x_s associated to input node inb^i is computed by the decoding algorithm by taking the majority of all the received messages in the same round as

$$M^{(d_{end},\nu_a)}_{inb^i \to outb^j} = \text{sign}\left(\sum_{b=1}^{n} \sum_{p=1}^{i} M^{(d_{end},\nu_b)}_{inb^i \leftarrow outb^{j'_p}} \right),$$

where $M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_1}}, \cdots, M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_{i-1}}}, M^{(d,\nu_b)}_{inb^i \leftarrow outb^{j'_i}}$ are messages related to all output nodes $outb^{j'_p}$, related to BSC(ν_b), adjacent to inb^i including $outb^j$, $p \in \mathfrak{I}_1^i$, $b \in \mathfrak{I}_1^n$, $j'_p \in \mathfrak{I}_1^k$ and d_{end} is the last round of the SP decoding.

Density Evolution

Density evolution is a tool to study the asymptotic evolution of the distribution of the messages during the iterative decoding. In this part, we derive the probability of sending a symbol $s \in \{-1, 0, 1\}$ along an edge in each

round of the algorithm. We use an LT code to transmit information via a PSM-BSC parameterized by $([\tau_1, \tau_2, \ldots, \tau_{n-1}], [\nu_1, \nu_2, \ldots, \nu_n])$ and respective fractional durations $d_1, d_2, \ldots, d_n$, where $\sum_{i=1}^{n} d_i = 1$. Due to the symmetry, we assume that the all $+1$ codeword has been transmitted. We expect that a $\sum_{i=1}^{n} d_i(1-\nu_i)$ fraction of the channel outputs are equal to $+1$, and a fraction $\sum_{i=1}^{n} d_i\nu_i$ of them are equal to -1. We use N output bits to decode the k information bits. The error correction will succeed if and only if all the input nodes send $+1$ to their neighbours after a large enough number of rounds. We use the following notation throughout.

- If $G(z) \triangleq \sum_{f=-\infty}^{+\infty} g_f z^f$, then $G(z)_{f<0} \triangleq \sum_{f<0} g_f$, $G(z)_{f=0} \triangleq g_0$ and $G(z)_{f>0} \triangleq \sum_{f>0} g_f$,
- $P^j(s)$ denotes the probability of sending s along an edge from an ouput node node related to BSC(ν_j) to an input node,
- $Q^j(s)$ denotes the probability of sending s along an edge from an input node to an output node related to BSC(ν_j),

where $s \in \{-1, 0, 1\}$ and $j \in \mathcal{I}_1^n$.
The density evolution calculations corresponding to each type of message are given as follows.

Algorithm 12. *Density Evolution of the Messages*

1. **Messages passed from output nodes to input nodes.**
   We have

$$P^j(-1) = \omega\Big(1 - Q^j(0)\Big)\left(\frac{1-(1-2\nu_j)\omega(1-2Q^j(0))}{2}\right),$$

$$P^j(+1) = \omega\Big(1 - Q^j(0)\Big)\left(\frac{1+(1-2\nu_j)\omega(1-2Q^j(0))}{2}\right),$$

$$P^j(0) = 1 - \omega\Big(Q^j(+1) + Q^j(-1)\Big).$$

2. **Messages passed from input nodes to output nodes.**
   We have

$$Q^j(-1) = F(z)^{(j)}_{f<0},$$

$$Q^j(+1) = F(z)^{(j)}_{f>0},$$

$$Q^j(0) = F(z)^{(j)}_{f=0}.$$

   where $F(z)^{(j)} = \sum_{d_j \geq 1} \iota^{(j)}_{d_j} [P^j(-1)z^{-1} + P^j(0) + P^j(+1)z]^{d_j}$
   $\prod_{h \in S_j} I^{(h)}_{d_h} [P^h(-1)z^{-1} + P^h(0) + P^h(+1)z]^{d_h}$
   and $S_j = \{h \neq j, d_h \geq 0, h \in \mathcal{I}_1^n\}$

Numerical Results

In Fig. 3.9, we present some simple examples to illustrate the behavior of the previous density evolution algorithm in terms of error probability versus the number of iterations. All results were obtained using the output degree distribution Ω [1] given by its generating function $\Omega(x) = 0.1629 + 0.3530x^2 + 0.0941x^3 + 0.0455x^4 + 0.0942x^5 + 0.097x^6 + 0.0154x^{11} + 0.0875x^{12} + 0.0004x^{87} + 0.05x^{88}$.

We assume all +1 codeword is sent. The decoder will correct the error if and only if the expression $\sum_{i=1}^{n} d_i Q^i(+1)$ converges to 1 after a bounded number of iterations. PSM-BSC($[0.25N], \nu_1, \nu_2$) refers to a PSM-BSC with the parameter $(\mathcal{T}, R) = (\{0.25N\}, [\nu_1, \nu_2])$, meaning that respective fractional durations are $d_1 = 0.25$ and $d_2 = 0.75$ for an assumed large value of N. We notice a convergence to an error floor after about 10 iterations.

3.2 Markov Modulated Channels

In the previous section, we provided SP decoding algorithms for a class of PSMC's. In this part, we give an estimation-decoding algorithm for LT codes over a class of channels that generalizes PSMC's making the channel state equal to the state of a hidden Markov chain.

3.2.1 Markov-modulated binary symmetric channels

A row vector of $k > 0$ bits is encoded using a specific LT code $x = (x_1, \cdots, x_k)$ into the row vector $z = (z_1, \cdots, z_N)$. This vector z is then transmitted over a particular markov-modulated binary symmetric channel, which the description is given below. We denote by $y = (y_1, \cdots, y_N)$ the received row vector and by $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$ an estimate of x obtained after the decoding.

Assume there exists a random variable $s \in \mathcal{S}$, called the *channel state*, where $\mathcal{S} = [s_1, s_2, \ldots, s_{|\mathcal{S}|}]$ is a discrete set of channel behaviors and let $V = [\nu_1, \nu_2, \ldots, \nu_{|\mathcal{S}|}]$ be the vector of parameters corresponding to these behaviors. Let a Markov chain $\mathcal{M}$ of length n, with a hidden vector $s = (s_1, \cdots, s_n) \in \mathcal{S}^n$. Consider a channel with random input $Y_t \in \{0, 1\}$, and random output $Z_t \in \{0, 1\}$, where $t \in \mathcal{I}_1^N$ $(N > 0)$, represents a time index. At each time t, the conditional probability function of Y_t given Z_t is a function of some parameter $s_a \in \mathcal{S}$ denoted by

$$\Pr[Y_t = y_t \mid Z_t = z_t; s_a] \triangleq \Pr[y_t \mid z_t; s_a],$$

where $a \in \mathcal{I}_1^{|\mathcal{S}|}$, $\mathcal{S}$ is the set of states of the hidden Markov chain $\mathcal{M}$ with a state transition probability matrix $\mathcal{P}$ and $|\mathcal{S}|$ its cardinality.

[1] Optimized degree distributions for transmission over a BSC(0.11) taken from [37].

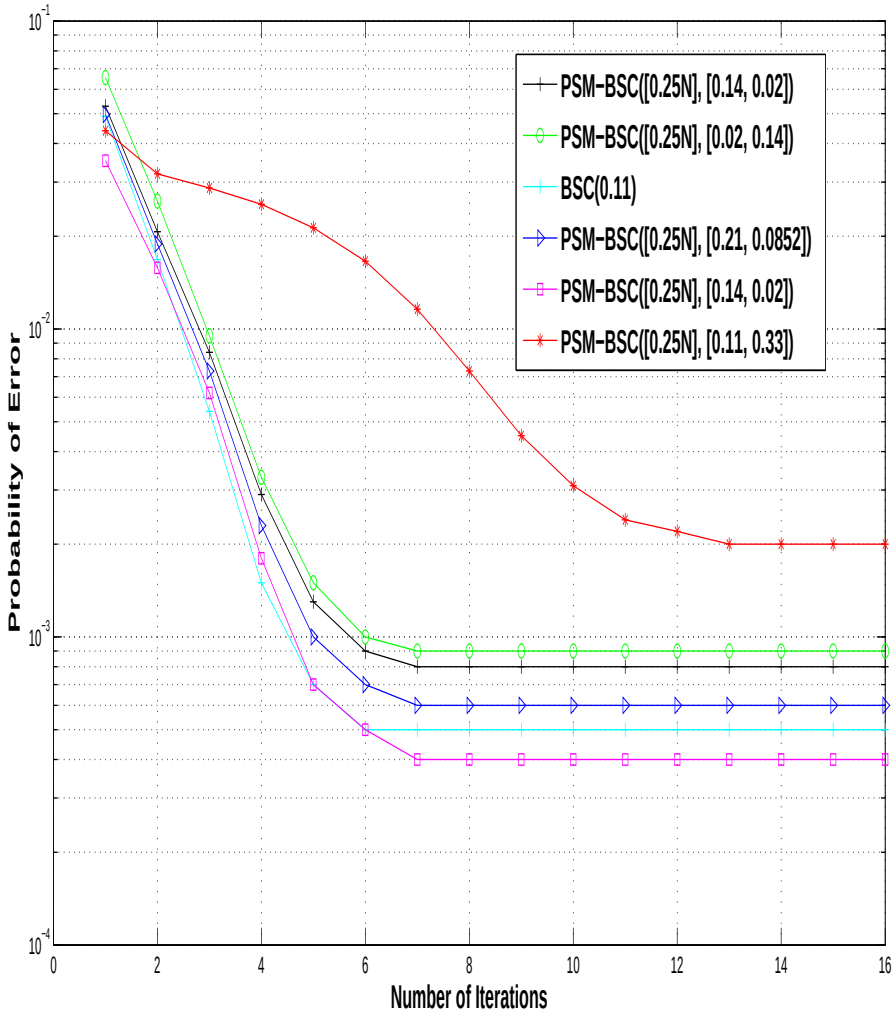


Figure 3.9: Error probability convergence using Gallager's majority decoding

If $b \in \mathcal{N}$ and $r_b \in \mathcal{S}$, a markov-modulated binary symmetric channel (MM-BSC) is a channel where $r = (r_1, \cdots, r_N)$, $z = (z_1, \cdots, z_N)$ and $y = (y_1, \cdots, y_N)$ have the following properties:

- There exists a Markov chain with a set of states $\mathcal{S}$ and a state transition probability matrix $\mathcal{P}$ such that the probability mass function of the channel state process can be written as

$$\Pr(r) = \Pr(r_1) \prod_{i=1}^{N} \Pr(r_{i+1}|r_i). \tag{3.7}$$

- The underlying channel represents a BSC(ν) family, parameterized by the crossover probability ν. If $r_b = s_a$, then the conditional channel

probability mass function is given by

$$\Pr[y_t \mid z_t; r_b] = \begin{cases} 1-\nu_a, & y_t = z_t, \\ \nu_a, & y_t \neq z_t. \end{cases}$$

- The channel state vector s is independent of the transmitted vector z (channel input vector) and the members of z are conditionally independent given the channel vector y. Thus if V is known, then

$$\Pr[y \mid z; r] = \Pr(r_1) \prod_{i=1}^{N} \Pr(r_{i+}|r_i) \Pr[y_i \mid z_i; r_i]. \tag{3.8}$$

We will concentrate on MM-BSC with two states called Gilbert-Elliott (GE) channel, with state alphabet $\mathcal{S} = [B, G]$, where the state labeled B refered sometimes to the bad state and G refered sometimes to the good state. States B and G are assigned respectively crossover probabilities ν_B and ν_G, so that

$$\Pr[y_t \neq z_t | s = r_b] = \begin{cases} \nu_B, & r_b = B, \\ \nu_G, & r_b = G. \end{cases}$$

The matrix $\mathcal{P}$ is given as

$$\mathcal{P} = \begin{pmatrix} 1-g & g \\ b & 1-b \end{pmatrix}.$$

The GE channel is parameterized by the 4-tuple (b, g, ν_B, ν_G), where g is the transition probability from state B to state G, b is the transition probability from state G to state B, $0 < b, g < 1$ and $0 \leq \nu_G < \nu_B \leq 1$.

3.2.2 SP based Estimation-Decoding Algorithms

Definition 6. *A factor graph ([29]) is a bipartite graph that represents the graphical structure of a factorization. A factor graph has a variable node for each variable v, a factor node for each local function f, and an edge-connecting variable node v to factor node f if and only if v is an argument of f.*

The factor graph representation of the LDPC codes over GE channels have been derived in [3]. Estimation-decoding of LT codes on GE channels can also be represented by a factor graph; we denote this factor graph as Markov-LT factor (see Fig. 3.10) graph. The Markov-LT factor graph can be divided into two subgraphs. The first part related to the Markov chain is the Markov subgraph (Fig. 3.11), where factor nodes and variables nodes refer respectively to channel factor nodes and state variable nodes. The second part is the LT subgraph, where factor nodes and variables nodes refer respectively to input and output nodes.
In what follows, it is assumed that the GE parameter (b, g, η_B, η_G) is perfectly

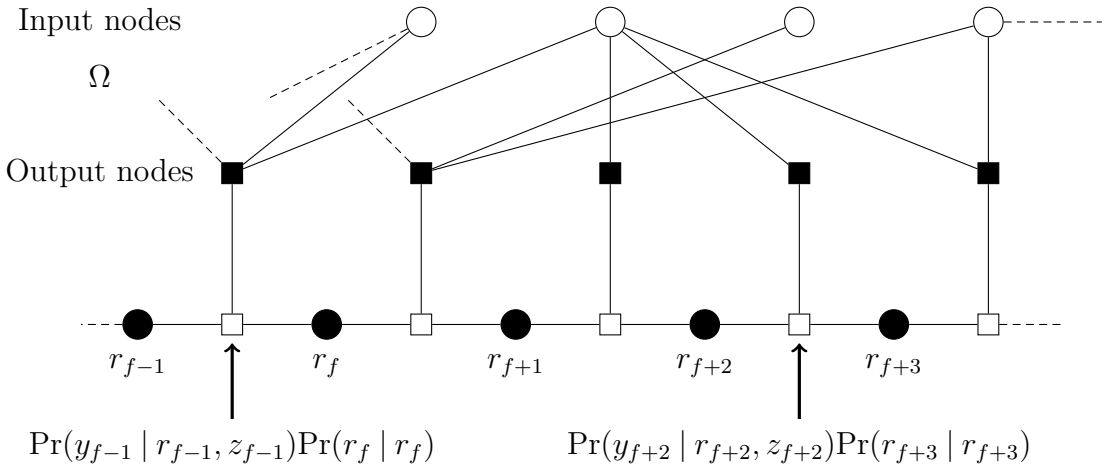


Figure 3.10: Markov-LT factor graph

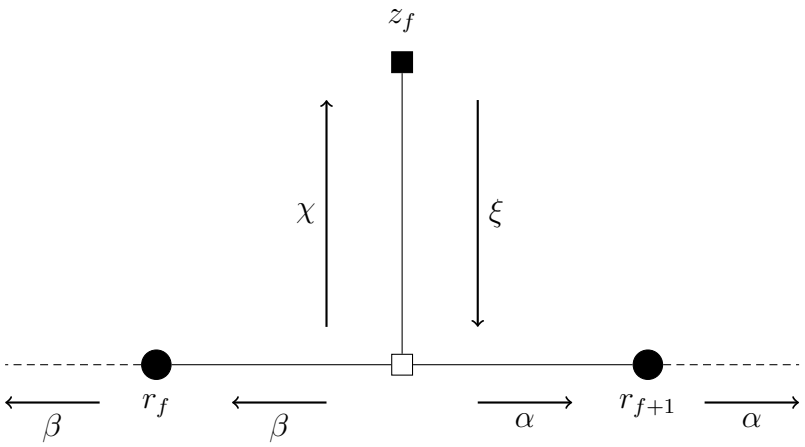


Figure 3.11: Message flow through the Markov-subgraph

known. We give a description of the SP algorithm that is used in the decoding process of LT codes over GE channels.

The SP decoding algorithm proceeds in rounds. At every round $d \geq 0$, messages are passed from input nodes to output nodes, and then from output nodes back to input nodes along the edges of a decoding graph for the given LT code.

Let us denote the input node of degree i by inb^i, the output node of degree j by $outb^j$, the message sent from input node inb^i to output node $outb^j$ at round d by $M^{(d)}_{inb^i \to outb^j}$, the message sent from output node $outb^j$ to input

node inb^i at round d by $M^{(d)}_{inb^i \leftarrow outb^j}$and

$$\xi \;=\; \Big(\xi_1, \cdots, \xi_N\Big)$$

as the LLR information row vector of the given GE channel, and

$$\chi \;=\; \Big(\chi_1, \cdots, \chi_N\Big)$$

as the extrinsic information row vector. The descriptions of ξ and χ are given below.

For $f \in \mathfrak{I}_1^N$, do

$$M^{(0)}_{inb^i \leftarrow outb^j} = \begin{cases} \xi_f & \text{if the degree of } outb^j \text{ is 1,} \\ 0 & \text{if the degree of } outb^j \text{ is bigger than 1.} \end{cases}$$

For $d \geq 0$, the SP update rules for the next steps are given as follows.

Algorithm 13. *Sum-Product decoding rule*
Input: Row vector of noisy output bits $y = (y_1, \cdots, y_N)$ and the corresponding graph for an LT code with parameters (k, Ω).
Output: Row vector of estimates of input bits $\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_k)$.

1. **LT-subgraph messages :**
In the LT subgraph (as described in Chapter 1), set the components of the message $M^{(d)}_{inb^i \to outb^j}$ from input node inb^i to output node $outb^j$ and those of the message $M^{(d+1)}_{inb^i \leftarrow outb^j}$ from output node $outb^j$ to input node inb^i as follows :

$$M^{(d)}_{inb^i \to outb^j} \;=\; \sum_{p=1}^{i-1} M^{(d)}_{inb^i \leftarrow outb^{j'_p}},$$

$$M^{(d+1)}_{inb^i \leftarrow outb^j} \;=\; 2\text{arctanh}\left(\tanh\left(\frac{\xi_f}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M^{(d)}_{inb^{i'_l} \to outb^j}}{2}\right)\right),$$

where the messages $M^{(d)}_{inb^i \leftarrow outb^{j'_1}}, \cdots, M^{(d)}_{inb^i \leftarrow outb^{j'_{i-1}}}$ are related to all output nodes $outb^{j'_p}$ adjacent to inb^i other than $outb^j$, $p \in \mathfrak{I}_1^{i-1}$, $j'_p \in \mathfrak{I}_1^k$; $M^{(d)}_{inb^{i'_1} \to outb^j}, \cdots, M^{(d)}_{inb^{i'_{j-1}} \to outb^j}$ are related to all input nodes $inb^{i'_l}$ adjacent to $outb^j$ other than inb^i, $l \in \mathfrak{I}_1^{j-1}$ and $i'_l \in \mathfrak{I}_1^N$.

2. **Markov-subgraph messages :**
In the Markov subgraph (see Fig. 3.11), we have two type of messages: forward messages and backward messages. For $i \in \mathfrak{I}_1^N$, we denote $\Pr(z_f|\chi_f)$ being the probabilistic form of χ_f given by

$$\Pr(z_f|\chi_f) = \begin{cases} \frac{1}{2} + \frac{1}{2}\tanh\frac{\chi_f}{2}, & z_f = 0; \\ \frac{1}{2} - \frac{1}{2}\tanh\frac{\chi_f}{2}, & z_f = 1; \end{cases}$$

where χ carries the belief message from an output symbol to a channel factor node, excluding the channel information. The forward message, represented by $\alpha = (\alpha(r_1), \cdots, \alpha(r_N))$, is passed from a channel factor node to a state variable node and carries channel state information from all previous channel observations.

$$\alpha^{(d+1)}(r_{f+1}) = \sum_{r_f \in \mathcal{S}} \Pr(r_{f+1} \mid r_f)\alpha^{(d)}(r_f) \sum_{z_f \in \{0,1\}} \Pr(z_f|\chi)\Pr(y_f \mid r_f, z_f),$$

where $r_{f+1} \in \mathcal{S}$.
The backward message, represented by $\beta = (\beta(r_1), \cdots, \beta(r_N))$, is passed from a channel factor node to a state variable node and carries channel state information from all future channel observations.

$$\beta^{(d+1)}(r_f) = \sum_{r_{f+1} \in \mathcal{S}} \Pr(r_{f+1}|r_f)\beta^{(d)}(r_{f+1}) \sum_{z_{f+1} \in \{0,1\}} \Pr(z_{f+1}|\chi)p(y_{f+1}|r_{f+1},$$

$z_{f+1})$, *where $r_f \in \mathcal{S}$.*

3. **Messages from LT-subgraph to Markov-subgraph :**
Messages sent from LT-subgraph to Markov-subgraph are denoted as extrinsic messages. The extrinsic message χ_f is passed from an output node $outb^j$ of degree d_j to a channel factor node; it's represented by χ.

$$\chi_f = 2\text{arctanh}\left(\prod_{l=1}^{j} \tanh\left(\frac{M^{(d)}_{inb^{i'_l} \to outb^j}}{2}\right)\right),$$

where $M^{(d)}_{inb^{i'_1} \to outb^j}, \cdots, M^{(d)}_{inb^{i'_j} \to outb^j}$ are related to all input nodes $inb^{i'_l}$ adjacent to $outb^j$ and $i'_l \in \mathfrak{I}_1^N$.

4. **Messages from Markov-subgraph to LT-subgraph :**
Messages sent from Markov-subgraph to LT-subgraph are denoted as channel messages. The channel message ξ_f is passed from channel factor to an ouput node; its carrries the belief of the value of the symbol, which corresponds to that output node, based on all available channel state information.

$$\xi_f = \log\left(\frac{\Pr(z_f = 0|\alpha, \beta)}{\Pr(z_f = 1|\alpha, \beta)}\right),$$

where

$$\Pr(z_f = 0|\alpha, \beta) = \sum_{r_f \in \mathcal{S}} \sum_{r_{f+1} \in \mathcal{S}} \Pr(y_f \mid r_f, z_f = 0)\Pr(r_{f+1} \mid r_f)\alpha(r_f)\beta(r_{f+1})$$

and $\Pr(z_f = 0|\alpha, \beta) = 1 - \Pr(z_f = 1|\alpha, \beta)$

The SP-based decoding algorithm proceeds in rounds. Every round proceeds in two steps. First the algorithm updatws messages at the LT-subgraph, and passes the appropriate messages to the Markov-subgraph. Then the algorithm updates messages at the Markov-subgraph, and passes the appropriate messages to the LT-subgraph.

Numerical Results

Fig. 3.12 and Fig. 3.13 illustrate the performance of the joint estimation-decoding (ED) method in comparison to the SP with the perfect knowledge (PK) of the GE channel statistics and the location of channels states G and B. Genie refers to the joint-decoding decoder where it's assumed that extrinsic messages are provided to the decoder.

All experimental results were obtained using the output degree distribution $\Omega(x) = 0.007690 + 0.493570x + 0.166220x^2 + 0.072646x^3 + 0.082558x^4 + 0.056058x^7 + 0.037229x^8 + 0.055590x^{18} + 0.025023x^{64} + 0.003135x^{65}$.
The GE channel is parameterized by the 4-tuple $(b, g, \nu_B, \nu_G) = (0.01, 0.1, 0.4, \nu_G)$. In Fig. 3.12, the values of k and N are chosen to be 22500 and 40000. In Fig. 3.13, the values of k and ν_G are chosen to be 22500 and 0.01. Generally speaking, as can be seen in the figures, the SP with the perfect location and knowledge (PK) of GE statistics perform better than our estimation-decoding algorithm. Moreover, our algorithm perform very close to the SP decoding with PK, as we provide extrinsic information to the decoder during decoding.

3.3 Conclusions

In this chapter, we considered implementations of the density evolution algorithms used in the decoding process of LT codes for stationary binary memoryless symmetric channels with assumed known channel statistics. We provided algorithms for joint LT estimation-decoding over piecewise stationary memoryless channels with a number of abrupt changes in channel statistics, considering binary symmetric channels with a crossover probability that changes a bounded number of times with no repetition in the statistics. We obtained that the decoding algorithm based on recursive segmentation performs almost as well as the decoding algorithm with the perfect knowledge of statistics. We also provided joint LT estimation-decoding over a family of Markov-modulated channels, called Gilbert-Elliot channels. For varying good state channel crossover ν_G , empirical results illustrated that our joint-decoding algorithm is close to the sum-product decoding where Gilbert-Elliot channels parameters and the location of channel states are assumed perfectly known, for large values of ν_G in terms of bit error probability.

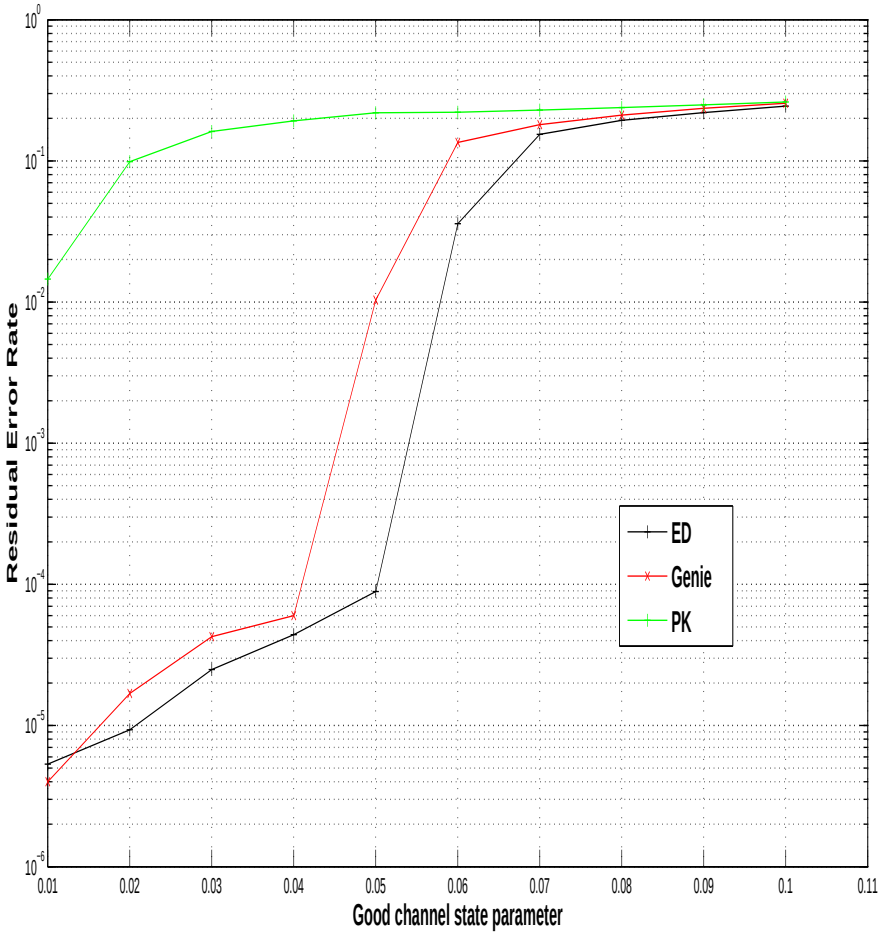


Figure 3.12: BER Performance comparison with varying good state channel crossover.

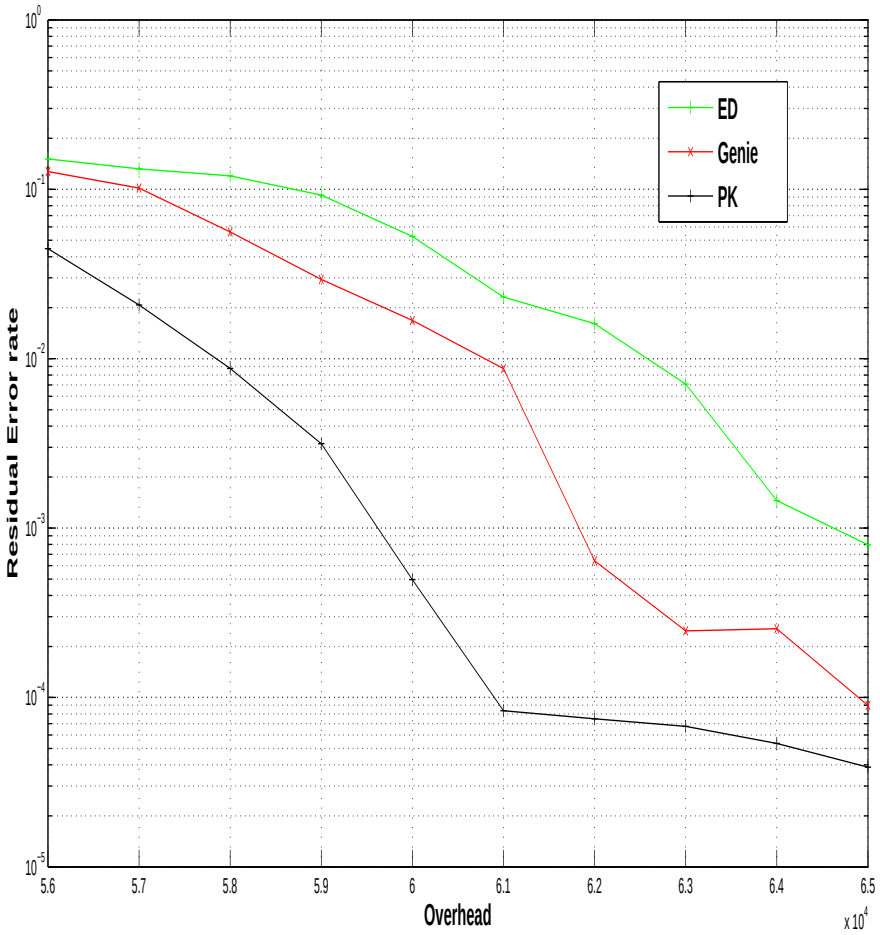


Figure 3.13: BER Performance comparison with varying overhead.

Systematic LT Codes for Lossless Coding

4

In this chapter, we address the problem of practical lossless source coding using systematic LT codes. In Section 4.1, we provide a compression algorithm by using a multilevel approach based on systematic LT codes. Our algorithm is based on the Burrows-Wheeler Transform, an invertible permutation transform that has been suggested for lossless data compression. The algorithm proceeds as follows. After applying the Burrows-Wheeler Transform, we reduce the number of symbols by applying a Run Length Encoding scheme. Then, we transform the local context of the symbols into a global context by the Incremental Frequency Count stage. At the end, we separately encode the run-length data stream with an Entropy Coder and the Incremental Frequency Count-index stream with a layered Fountain Coder. The proposed scheme follows the Closed-Loop Iterative Doping algorithm together with the multilevel stage decoding Sum-Product at the Frequency Count stage. Our algorithm offers encouraging compression rate performance for large files. In Section 4.2, we describe one solution to the two-user Slepian-Wolf problem in a certain part of the achievable region using fountain codes. Symmetric case of memoryless compression of two correlated sources is considered and modeled by binary symmetric channels. The compression is done by two separate compressors without any exchange of information between them. The decompressor uses the Sum-Product decoding algorithm in conjunction with the Blind Iterative Doping strategy. Simulation results indicate performance close to the Slepian-Wolf limit.

4.1 Burrows-Wheeler Text Compression

4.1.1 Motivation

The Burrows-Wheeler compression algorithm[35] (BWCA) achieves strong compression rates and a high throughput. In contrast to many other compression approaches, the BWCA is a block oriented compression algorithm. A file to be compressed is first divided into data blocks of a fixed size and all blocks are processed separately. The size of a block is in the range of 1 to 10 MB in general. Since each block is processed separately, no context information of the previous block is used in the following blocks. For files larger than the block size, this is a disadvantage compared to streaming compression algorithms, which are able to exploit the context of the whole file.

The BWCA itself consists of several stages, which are performed sequentially. Each stage transforms the symbols of an input buffer into symbols of an output buffer, which is used as the input buffer for the next stage. A basic BWCA consists of three stages, one of which is the so called Burrows-Wheeler Transformation (BWT) stage, the second is the Global Structure Transformation (GST) stage, and the third is the Entropy Coding (EC) stage [41]. An introduction into BWCA is given by Fenwick in [42].

Consider a q-ary source S with alphabet $A = \{A_0, \dots A_{q-1}\}$ with cardinality $|A|$, where $q = 2^d$ and $d > 0$.

The first stage, the BWT, performs a permutation of the input symbols, which is the basis for the next stages. The symbols are reordered according to their context. The output of the BWT stage contains many runs of repeated symbols. During the last years, many fast and efficient algorithms for the BWT have been presented by Larsson and Sadakane [24], Sadakane [25], Itoh and Tanaka [20], Kao [21] and Kärkkäinen and Sanders [23].
We assume that the BWT has in input sequence $bwt_{in} = (bwt_{in,1}, \cdots, bwt_{in,N})$ and an ouput sequence $bwt_{out} = (bwt_{out,1}, \cdots, bwt_{out,n})$, where $N > 0$, $bwt_{in,i}, bwt_{out,i} \in A$ and $i \in \mathcal{I}_1^n$. the BWT works in three steps.

Algorithm 14. *BWT*
Input: Row vector of symbols $bwt_{in} = (bwt_{in,1}, \cdots, bwt_{in,N})$.
Output: Row vector of symbols $bwt_{out} = (bwt_{out,1}, \cdots, bwt_{out,N})$ *and an index* I*, which we described below.*

1. *Reverse the sequence* bwt_{in} *and then obtain the sequence* $c = (c_1, \cdots, c_N)$*, where* $c_j = bwt_{out,n-j+1}$ *and* $j \in \mathcal{I}_1^n$.

2. *Compute* N *cyclic shifted versions of* c*, and sort the cyclic shifts.*

3. *Output the last column of the sorted cyclic shifts along with the index* I *of the cyclic shift corresponding to the sequence* c.

Cyclic shifts
version_1: a, l, g, o, m, a, t, h
version_2: h, a, l, g, o, m, a, t
version_3: t, h, a, l, g, o, m, a
version_4: a, t, h, a, l, g, o, m
version_5: m, a, t, h, a, l, g, o
version_6: o, m, a, t, h, a, l, g
version_7: g, o, m, a, t, h, a, l
version_8: l, g, o, m, a, t, h, a

Table 4.1: BWT: Cyclic shifts

Sorted shifts
version_1: a, l, g, o, m, a, t, h
version_4: a, t, h, a, l, g, o, m
version_7: g, o, m, a, t, h, a, l
version_2: h, a, l, g, o, m, a, t
version_8: l, g, o, m, a, t, h, a
version_5: m, a, t, h, a, l, g, o
version_6: o, m, a, t, h, a, l, g
version_3: t, h, a, l, g, o, m, a

Table 4.2: BWT: Sorting

Example 8. *We compute the BWT of* $bwt_{in} = (h, t, a, m, o, g, l, a)$.
We reverse the input to form $C = (a, l, g, o, m, a, t, h)$ *and compute the cyclic shifts and sorted shifts as in Tab. 4.1 in Tab. 4.2.*

The last column of the sorted matrix is (h, m, l, t, a, o, g, a)*, and c is in row* 1 *in this matrix.* $BWT(bwt_{in}) = [(h, m, l, t, a, o, g, a), 1]$. ■

The second stage of the BWCA transforms the local structure of the BWT output stream into an index stream with a global structure and is called a Global Structure Transformation (GST).The most common approach is the fast Move To Front (MTF) stage, which was used in the original scheme by Burrows and Wheeler [35].
The MTF converts a sequence of characters to a list of numbers. A sequence $mtf_{in} = (mtf_{in,1}, \cdots, mtf_{in,N})$ is encoded to a list of numbers $mtf_{out} = (mtf_{out,1}, \cdots, mtf_{iout,N})$ using the MTF as follows.

Algorithm 15. *MTF*
Input: Row vector of symbols $mtf_{in} = (mtf_{in,1}, \cdots, mtf_{in,N})$.
Output: Row vector of numbers $mtf_{out} = (mtf_{out,1}, \cdots, mtf_{iout,N})$.

1. *Maintain a list* $L = (L_1, \cdots, L_N)$ *of the* N *symbols in the alphabet* A.

2. For each symbol in $mtf_{in,i}$,

- *Encode the position L_i of this symbol in the list L.*
- *Modify the list by moving the symbol to the front of the list L..*

Example 9. *Assume $A = \{o, p, q, r\}$. We compute the MTF of $mtf_{in} = (q, o, r, o, o, p)$ and obtain $mtf_{out} = (3, 2, 4, 2, 1, 4)$* ■

Better compression rates are achieved by the more complex Weighted Frequency Count (WFC) stage from Deorowicz [40] and the Incremental Frequency Count (IFC) stage by Abel [22].

The IFC is one of several stages inside the BWCA. In our scheme, it is located between the BWT at the beginning of the algorithm and the FC at the end of the algorithm (Fig. 4.1).

For higher speed and better compression rate, a Run Lengh Encoder (RLE) stage is located directly in front of the IFC stage. The BWT places symbols with a similar context close together and produces many runs of repeated symbols. The RLE stage replaces all runs, which have a length of two or more symbols, by a run consisting of exactly two symbols. The length of a run is transmitted into a separate data stream (as indicated in Fig. 4.1) with an EC [41]. Then, the length information does not disturb the context of the main data stream. The basic idea of the IFC stage is the use of a rising increment and of counters for each alphabet symbol, similar to the counters of an arithmetic coder, which uses a fixed increment instead. The counters for the alphabet symbols form a counter list. The counter of the current symbol is increased by the increment and the counter list is resorted descendingly so that the symbol with the highest counter stands at the front of the list. Then, the increment is increased by the average value of the index values of the near past. Since only one counter is changed for each symbol processed, only one element needs to be resorted inside the counter list. This leads to an implementation, which is much faster than the WFC [14] GST from Deorowicz while still achieving strong compression rates. Similar to arithmetic coding, the increment and counters are halved if a counter exceeds a fixed threshold. The compression rates of the IFC stage are in the range of the results of the WFC stage, while the speed is similar to the MTF stage.

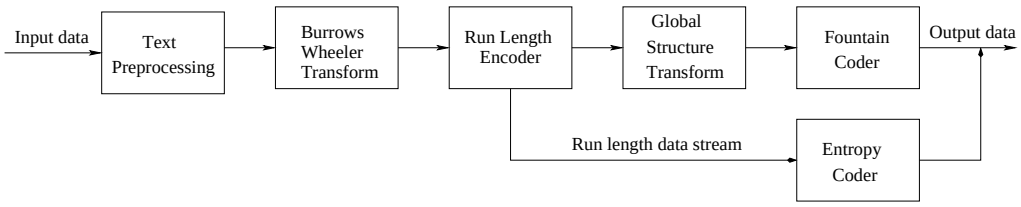


Figure 4.1: Source coding scheme for text compression

4.1.2 Text Preprocessing

The Text Preprocessing (TP) proposed [2] consists of six separate algorithms: text categorizing, capital letter conversion, space stuffing, word replacement, phrase replacement and alphabet reordering. The algorithms need no fixed external information like dictionaries and are language independent. The different techniques are processed sequentially in order to complement and boost each other. We note a compression improvement higher than the sum of the improvements of the separate algorithms if all these algorithms are performed behind each other. In what follows, we describe each algorithm of the TP used in our text compression algorithm.

Text categorizing: Since TP hampers the compression rate of non text files, it is important to recognize a file as a text file and to leave other files unchanged. Here a simple but efficient algorithm is used [2]. In order for a file to be recognized as a text file and to be preprocessed by the following algorithms, it has to fulfill two conditions:

1. The percentage frequency share of alphanumerical symbols A $\cdots$ Z, a $\cdots$ z, $0 \cdots 9$,' ' compared to all symbols must be greater than 66 percent.

2. The percentage frequency share of the space symbol compared to alphanumerical symbols must be greater than 10 percent.

The result of the categorizing is saved as the first byte in the output stream followed either by the preprocessed text or by the unchanged data, in case the file did not fulfill the two conditions.

Capital letter conversion: The basic idea of capital letter conversion is to replace words, which begin sometimes with a capital letter and sometimes with a lowercase letter, always by a starting lowercase letter in order to produce a more stable context. Capital letter conversion achieves a compression improvement around 0.5 percent on the text files of the Calgary corpus. The algorithm consists of two loops. In the first loop, all words of the text with length 2 or more are saved in a ternary search tree. In the second loop, words, which start with a capital letter and that occur at least once with a lowercase letter inside the text and that second letter is a lowercase letter, are replaced by a capital-letter escape symbol, followed by a space symbol and the corresponding lowercase starting letter.

Space stuffing: The space stuffing algorithm used in this approach adds a space symbol after each of the following symbols: '>', '(', ""'. If the current symbol is a TAB-symbol, space stuffing adds a space before and after the TAB-symbol. Although the total length of the file is enlarged, the context of the words is enhanced. The improvement achieved by this technique is only small and usually less than 0.1 percent.

Word replacement: Word replacement produces the highest compression improvement on average, typically between 1 and 2 percent on the text

files of the Calgary corpus[1]. This scheme replaces frequently used words by byte tokens. The words to replace are chosen by calculating the frequencies of all words of the text with length of size 2 or more using a ternary search tree. These words are weighted by their length and frequency. The words with the highest weight are replaced by tokens. The tokens are indices into a word dictionary. The dictionary is built adaptively and transmitted together with the data stream in order to keep the approach language independent by writing out the word the first time it appears. The size of the dictionary is defined by the number of tokens available, i.e., the number of unused symbols in the alphabet.

Phrase replacement: Besides word replacement, two most frequent bigrams and trigrams are also replaced by tokens. Therefore, four available tokens are reserved for phrase replacement. Using larger n-grams (where $n \geq 2$) than bigrams or trigrams does not lead to better compression rates in general, since they are much less frequent than bigrams and trigrams. The compression improvement is quite moderate with around 0.1 percent.

Alphabet reordering: For compression schemes based on sorting stages like the BWCA [35], the lexicographic order of the alphabet symbols has an influence on the output sequence. If the lexicographic order of the alphabet symbols is changed so that symbols with a similar context are grouped closer together, segments with similar context properties will also be grouped closer together, resulting in less context changes. In the present implementation, an alphabet order is used, which groups the vowels "aoui" in the middle of the consonants together and the 'e' at the end of the consonants. The complete alphabet order is presented in [2]. For BWCAs, alphabet reordering boosts the compression rate typically by about 1 percent on the text files of the Calgary corpus.

The probability distribution of a source output X is assumed to be $\Pr(X = a) = \Pr(a)$, for all a in A. Let $B(a) = (b_1(a), \ldots, b_\tau(a), \ldots, b_d(a))$ be the binary representation of the symbol a in A.

Definition 7. *We define the conditional marginal probability at level $\tau \in \mathcal{I}_1^d$ as*

$$\begin{aligned} \Pr_\tau(b_1, \ldots, b_{\tau-1}) &= \Pr(b_\tau(Y) = 1 | b_1(Y) = b_1, \ldots, b_{\tau-1}(Y) = b_{\tau-1}) \\ &= \frac{\sum_{y \in A: b_1(y) = b_1, \ldots, b_{\tau-1}(y) = b_{\tau-1}, b_\tau(y) = 1} \Pr(y)}{\sum_{y \in A: b_1(y) = b_1, \ldots, b_{\tau-1}(y) = b_{\tau-1}} \Pr(y)}. \end{aligned}$$

Definition 8. *We denote the conditional entropy of Y at level $\tau \in \mathcal{I}_1^d$ given its previous values at respective levels $1, \cdots, \tau - 1$ as*

$$H(b_\tau(Y) | b_1(Y), \ldots, b_{\tau-1}(Y)).$$

[1]The Calgary Corpus is a collection of files, commonly used for comparing compression schemes. It was created by Ian Witten, Tim Bell and John Cleary from the University of Calgary in 1987. In 1997, it was replaced by the Canterbury Corpus [1].

4.1.3 Modeling

Assume that the source sequence $y = (y_1, \ldots, y_n)$ takes values in a given q-ary alphabet A. If the sources are i.i.d., we can estimate the probability distributions empirically and plug the estimates into the entropy formula. If the sources are piecewise i.i.d., the BWT maps the source output of a stationary ergodic tree into a sequence that can be decomposed asymptotically into piecewise independent and identically i.i.d segments [26] as mentioned before. The first step of statistical modeling for the source consists of finding an efficient way of segmenting the source, and thus estimating the first order distribution on each segment, and finally estimating the empirical entropy of our source.

A source statistics model is mainly given by the number of segments, the distinct transition points between segments and by the model segment distributions. The cost of such a statistical model to represent y is the total number of bits needed to describe y. In [7, 6], to find the most efficient piecewise i.i.d. source model, the authors implemented a segment merging algorithm. This algorithm (explained in [10]) uses the MDL principle that learns an approximation to the source tree, identifying segments by their context. Cai et al. [5] used different approaches to approximate the source tree and proposed two different segmentation models first appeared in [45] and [44].

The first one called the adaptive segmentation estimates the location of the transitions of the BWT output sequence based on the empirical distribution of the symbols. This adaptive algorithm first obtains rough estimates for transition locations, and refines the locations of the estimates at the second pass. The numbers and lengths of the segments are adapted to the realization of the source, and this results in segmentations of different lengths.

The second approach is the uniform segmentation, in which the BWT output is partitioned so that each segment contains an equal number of symbols from the sequence according to which segmentation is done. We denote this number of symbols by $w(n)$. By taking $w(n)$ as $O(\sqrt{n})$, it has been established that, as n tends to $+\infty$, the entropy estimator converges to the entropy rate with high probability for stationary ergodic sources [5]. From experimental results, it has been established that the uniform segmentation method performs almost as well as the the adaptive method [5]. We will follow the uniform segmentation. The advantage of using this segmentation is that the encoder does not need to send the transition points between segments to the decoder. The decoder needs only to know the number of segments.

The source modeling algorithm works as follows:

1. Apply BWT on the sequence, followed by RLE. The RLE stage replaces all runs of repeated symbols length of two or more symbols, by a run consisting of exactly two symbols. The output of the RLE consists of two separate data streams, the run-symbols data stream (RSDS), and the run-length data stream (RLDS).

2. Apply a GST on the RSDS.

3. For each binary representation level τ from 1 to d,

   a) Let $\Gamma_\tau = \{1, \ldots, |\Gamma_\tau|\}$, where $|\Gamma_\tau|$ is the number of segments. Partition the BWT-RSDS-GST output sequence into $|\Gamma_\tau| = \frac{N}{w(N)}$ segments, with $w(N) = c_\tau\sqrt{N}$, where c_τ is a non negative number[2].

   b) Estimate the first order distribution within each segment. We estimate the number of occurrences of symbol y in the k-th segment by $N_k(y)$, and the probability estimate of symbol y in the k-th segment by $\Pr^k(y)$, as $\Pr^k(y) = \frac{N_k(y)+\beta}{\sum_{z\in A}(N_k(z)+\beta)}$, where β a small non negative number[3].

   c) Compute the contribution to the conditional entropy estimate of the empirical distribution in the k-th segment as

$$C_k = \sum_{z\in A}(N_k(z) + \beta)\log_2 \Pr{}^k_\tau(b_1, \ldots, b_{\tau-1}),$$

   with $\Pr^k_\tau(b_1, \ldots, b_{\tau-1})$ defined as the conditional marginal probability on the k-th segment, with $\Pr(y) = \Pr^k(y)$.
   Average the individual estimates. The estimate conditional entropy is

$$H(b_\tau(Y)|b_1(Y), \ldots, b_{\tau-1}(Y)) = \frac{-\sum_{k\in\Gamma_\tau} C_k}{n}.$$

   d) Compute the conditional probability Log-Likelihood Ratio (LLR) for each symbol y_j as $\log_2(\frac{1-\Pr^k_\tau(b_1(y_j),\ldots,b_{\tau-1}(y_j))}{\Pr^k_\tau(b_1(y_j),\ldots,b_{\tau-1}(y_j))})$.

4. Estimate the entropy as $H(Y) = \sum_\tau^d H(b_\tau(Y)|b_1(Y), \ldots, b_{\tau-1}(Y))$.

4.1.4 Encoder

In this part, the encoding part of the FC is described. The main building block of the source coding scheme using Fountain Codes outlined before for non binary Markov sources [6] is the main idea here. Let $\psi = \{\Omega^1, \ldots, \Omega^{|\Psi|}\}$ be a finite ensemble of LT code distributions optimized for erasure channels [17] and $(y_1, \ldots, y_N)$ the BWT-RSDS-GST output sequence. Let the sequence $(b_\tau(y_1), \ldots, b_\tau(y_N))$ be the binary representation of the sequence $(y_1, \ldots, y_N)$ at the τ-th binary representation level. Then the encoding proceeds as follows.

For each distribution Ω^s of Ψ and each binary representation level τ from 1 to d:

[2]Here c_τ is a design parameter of the algorithm which depends of the binary representation level and the length of the sequence.

[3]If the probability estimate in the segment k is 0, the logarithm of the probability tends to $-\infty$. The number β is introduced in order to deal with this issue.

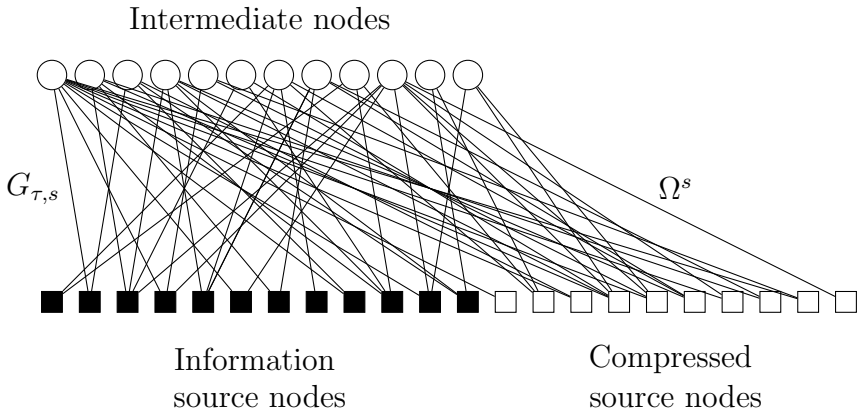
1. Calculate from row vector of binary symbols $(b_\tau(y_1), \ldots, b_\tau(y_N))$ a row vector of binary intermediate symbols $(z_{\tau,1}, \ldots, z_{\tau,N})$ through a linear invertible $N \times N$ matrix $G_{\tau,s}$ [7, 6]:

$$(z_{\tau,1}, \ldots, z_{\tau,N}) = G_{\tau,s}^{-1} \cdot (b_\tau(y_1), \ldots, b_\tau(y_N)).$$

   $G_{\tau,s}$ is obtained using Algo. 2 (see Chapter 1).

2. Generate m_s symbols [4] $(y_{\tau,(N+1)}, \ldots, y_{\tau,(N+m_s)})$ from $(z_{\tau,1}, \ldots, z_{\tau,N})$ through encoding with an LT-code with parameters (n, Ω^s), using Algo. 2 in Chapter 1.

3. A bipartite graph is set up between the nodes corresponding to the row vector of intermediate bits $(z_{\tau,1}, \ldots, z_{\tau,N})$ and on the other side the nodes corresponding to row vector of source bits $(b_\tau(y_1), \ldots, b_\tau(y_N))$ and the nodes corresponding to row vector of compressed bits $(y_{\tau,(N+1)}, \ldots, y_{\tau,(N+m_{\tau,s})})$ obtained previously, where $m_{\tau,s} = min(m_1, \ldots, m_{|\Psi|})$.
   Nodes corresponding to intermediates bits are called intermediate nodes; those corresponding to source bits are called information source nodes and those corresponding to compressed bits are compressed source nodes as graphically illustrated below.



4. The SP decoding algorithm is applied to this graph. The objective of the SP algorithm is to decode the symbols $(z_{\tau,1}, \ldots, z_{\tau,N})$ using the full knowledge of the symbols $(y_{\tau,(N+1)}, \ldots, y_{\tau,(n+m_{\tau,s})})$ and the absolute values of LLRs coming from the modeling part.
   Let us denote the intermediate node of degree i by $intb^i$, the information source node of degree j by $sourb^j$, the compressed source node of degree k by $comprb^k$ the message sent from intermediate node $intb^i$ to information source node $sourb^j$ at round d by $M^{(d)}_{intb^i \to sourb^j}$, the message

[4] These m_s output symbols, together with the doped symbols obtained from the CLID algorithm constitute the output of the compressor; m_s should be as close as possible to $n(H(b_\tau(Y)/b_1(Y), \ldots, b_{\tau-1}(Y)) + \phi)$, where ϕ is a small non negative bias.

sent from intermediate node $intb^i$ to compressed source node $comprb^k$ at round d by $M^{(d)}_{intb^i \to comprb^k}$, the message sent from information source node $sourb^j$ to intermediate node $intb^i$ at round d by $M^{(d)}_{intb^i \leftarrow sourb^j}$ and the message sent from compressed source node $comprb^k$ to intermediate node $intb^i$ at round d by $M^{(d)}_{intb^i \leftarrow compr^k}$.
The initial LLR of $(z_{\tau,1}, \ldots, z_{\tau,n})$, $(b_\tau(y_1), \ldots, b_\tau(y_N))$ and $(y_{\tau,(n+1)}, \ldots, y_{\tau,(n+m_{\tau,s})})$ are 0, $LLR_{r,\tau}|\log_2(\frac{1-\Pr^k_\tau(b_1(y_r),\ldots,b_{\tau-1}(y_r))}{\Pr^k_\tau(b_1(y_r),\ldots,b_{\tau-1}(y_r))})|$ and $LLR_u = +\infty$, where $r \in \mathcal{I}_1^N, u \in \mathcal{I}_1^{m_{\tau,s}}$.

For $j, s, i, r, \in \mathcal{I}_1^N$ and $k, u \in \mathcal{I}_1^{m_{\tau,s}}$, we assume that $sourb^j = sourb^{j,r}$ is associated to the variable y_r, $intb^i = intb^{i,s,\tau}$ is associated to the variable $z_{\tau,s}$ and $comprb^k = comprb^{k,u}$ to the variable y_u. In the very first round, information source nodes and compressed source nodes with degree 1, send their values to their unique neighbours in the set of the input nodes, which can be written as

For $r \in \mathcal{I}_1^N$, do

$$M^{(0)}_{intb^i \leftarrow sourb^j} = \begin{cases} LLR_{r,\tau} & \text{if the degree of } sourb^j \text{ is 1,} \\ 0 & \text{if the degree of } sourb^j \text{ is bigger than 1,} \end{cases}$$

and for $r \in \mathcal{I}_1^{m_{\tau,s}}$, do

$$M^{(0)}_{intb^i \leftarrow comprb^k} = \begin{cases} +\infty & \text{if the degree of } comprb^k \text{ is 1,} \\ 0 & \text{if the degree of } comprb^k \text{ is bigger than 1} \end{cases}$$

For $d \geq 0$, the SP update rules for the next steps are given as follows.

a) **Messages from intermediate nodes to information source nodes.**
Set the components of the message $M^{(d)}_{intb^i \to \text{sourb}^j}$ from intermediate node $intb^i$ to information source node $sourb^j$ as follows.
For $s \in \mathcal{I}_1^N$, do

$$M^{(d)}_{intb^i \to \text{sourb}^j} = \sum_{p=1}^{i_1-1} M^{(d)}_{int^i \leftarrow sourb^{j'_p}} + \sum_{p'=1}^{i_2} M^{(d)}_{intb^i \leftarrow comprb^{j'_{p'}}}$$

where the messages $M^{(d)}_{intb^i \leftarrow comprb^{j'_1}}, \cdots, M^{(d)}_{inbt^i \leftarrow comprb^{j'_2}}$ are related to all compressed source nodes $comprb^{j'_{p'}}$, $M^{(d)}_{intb^i \leftarrow sourb^{j'_1}}, \cdots,$ $M^{(d)}_{inbt^i \leftarrow sourb^{j'_{i-1}}}$ are related to all information source nodes $sourb^{j'_p}$ adjacent to $intb^i$ other than $sourb^j$, $p \in \mathcal{I}_1^{i_1-1}, p' \in \mathcal{I}_1^{i_2}$, $j'_p, j'_{p'} \in \mathcal{I}_1^N$, $i_1 > 0, i_2 > 0$ and $i_1 + i_2 = i$.

b) **Messages from intermediate nodes to compressed source nodes.**

Set the components of the message $M^{(d)}_{intb^i \to \mathrm{comprb}^k}$ from intermediate node $intb^i$ to compressed source node $comprb^k$ as follows.
For $s \in \mathcal{I}_1^N$, do

$$M^{(d)}_{intb^i \to \mathrm{comprb}^j} = \sum_{p=1}^{k_1-1} M^{(d)}_{int^i \leftarrow comprb^{j'_p}} + \sum_{p'=1}^{k_2} M^{(d)}_{intb^i \leftarrow sourb^{j'_{p'}}}$$

where the messages $M^{(d)}_{intb^i \leftarrow sourb^{j'_1}}, \cdots, M^{(d)}_{inbt^i \leftarrow sourb^{j'_2}}$ are related to all information source nodes $sourb^{j'_{p'}}$ adjacent to $intb^i$, $M^{(d)}_{intb^i \leftarrow comprb^{j'_1}}$, $\cdots, M^{(d)}_{inbt^i \leftarrow comprb^{j'_{i-1}}}$ are related to all compressed source nodes $comprb^{j'_p}$ adjacent to $intb^i$ other than $comprb^j$, $p \in \mathcal{I}_1^{k_1-1}, p' \in \mathcal{I}_1^{k_2}$, $j'_p, j'_{p'} \in \mathcal{I}_1^N$, $k_1 > 0, k_2 > 0$ and $k_1 + k_2 = i$.

c) **Messages from source nodes to intermediate nodes.**
Set the components of the message $M^{(d+1)}_{inttb^i \leftarrow sourb^j}$ from information source node $sourb^j$ to intermediate node $intb^i$ as follows.
For $r \in \mathcal{I}_1^N$, do

$$M^{(d+1)}_{intb^i \leftarrow sourb^j} = 2\mathrm{arctanh}\left(\tanh\left(\frac{LLR_{r,\tau}}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M^{(d)}_{intb^{i'_l} \to \mathrm{sourb}^j}}{2}\right)\right)$$

where the messages $M^{(d)}_{intb^{i'_1} \to sourb^j}, \cdots, M^{(d)}_{intb^{i'_{j-1}} \to sourb^j}$ are related to all intermediate nodes $intb^{i'_l}$ adjacent to $sourb^j$ other than $intb^i$, $l \in \mathcal{I}_1^{j-1}$ and $i'_l \in \mathcal{I}_1^N$.

d) **Messages from compressed source nodes to intermediate nodes.**
Set the components of the message $M^{(d+1)}_{inttb^i \leftarrow comprb^j}$ from compressed source node $comprb^j$ to intermediate node $intb^i$ as follows.
For $u \in \mathcal{I}_1^{m_{\tau,s}}$, do

$$M^{(d+1)}_{intb^i \leftarrow comprb^j} = 2\mathrm{arctanh}\left(\tanh\left(\frac{LLR_u}{2}\right).\prod_{l=1}^{j-1}\tanh\left(\frac{M^{(d)}_{intb^{i'_l} \to \mathrm{comprb}^j}}{2}\right)\right)$$

where the messages $M^{(d)}_{intb^{i'_1} \to comprb^j}, \cdots, M^{(d)}_{intb^{i'_{j-1}} \to comprb^j}$ are related to all intermediate nodes $intb^{i'_l}$ adjacent to $comprb^j$ other than $intb^i$, $l \in \mathcal{I}_1^{j-1}$ and $i'_l \in \mathcal{I}_1^{m_{\tau,s}}$.

5. During the SP decoding algorithm, the Closed-Loop Iterative Doping [8] (CLID) algorithm is applied. Every f-th [5] round of the iterations the

[5] Note that f is a design parameter of the algorithm.

intermediate bit with a smallest reliability is marked, and its LLR is set to $+\infty$ or $-\infty$, depending on whether its value is 0 or 1. The precise description of the CLID is as follows. For $d_f = f, 2f, 3f, \cdots$ iterations,

- For all intermediate nodes, compute

$$M^{d_f}_{intb^{i,s,\tau}\to \mathrm{sourb}^j} = \sum_{p=1}^{i_1} M^{d_f}_{int^{i,s,\tau}\leftarrow sourb^{j'_p}} + \sum_{p'=1}^{i_2} M^{d_f}_{intb^{i,s,\tau}\leftarrow comprb^{j'_{p'}}}$$

  where the messages $M^{(d)}_{intb^i\leftarrow comprb^{j'_1}}, \cdots,$ $M^{(d)}_{inbt^i\leftarrow comprb^{j'_2}}$ are related to all compressed source nodes $comprb^{j'_{p'}}$ adjacent to $intb^i$, $M^{(d)}_{intb^i\leftarrow sourb^{j'_1}},$ $\cdots, M^{(d)}_{inbt^i\leftarrow sourb^{j'_{i-1}}}$ are related to all information source nodes $sourb^{j'_p}$ adjacent to $intb^i$, $p \in \mathfrak{I}_1^{i_1-1}, p' \in \mathfrak{I}_1^{i_2}$, $j'_p, j'_{p'} \in \mathfrak{I}_1^N$, $i_1 > 0, i_2 > 0$, $i_1 + i_2 = i$.
- For $s \in \mathfrak{I}_1^N$, sort the values $|M^{d_f}_{intb^{i,s,\tau}\to \mathrm{sourb}^j}|$ in increasing order.
- Feed the symbol $z_{\tau,opt}$ to the decoder, where

$$opt = \arg_{z_{\tau,s}, s\in\mathfrak{I}_1^N} \min\{|M^{d_f}_{intb^{i,s,\tau}\to \mathrm{sourb}^j}|\}$$

  The SP decoding decoding together with CLID continues until the intermediate symbols can be decoded from the LLRs of $(b_\tau(y_1), \ldots, b_\tau(y_N))$ and the output $(y_{\tau,(N+1)}, \ldots, y_{\tau,(N+m_s)})$.

6. In addition to the sequence $(y_{\tau,(N+1)}, \ldots, y_{\tau,(N+m_{\tau,s})})$, the encoder sends the number of segments $|\Gamma_\tau|$ and corresponding LLR values, the code distribution $\Omega^{\tau,s}$ of Ψ for which the number of doped symbols is smallest, a seed for the matrix construction of $G_{\tau,s}$ to the decoder using an *adaptive Huffman* [41] coding.

4.1.5 Decoder

In this section, the decoding steps of the proposed scheme are described. The decoder works as follows.

1. At the inverse FC part, the decompression is achieved level by level in several steps which closely mimic the compression steps. For each binary representation level τ from 1 to d, using the *adaptive Huffman* [41] decoding, the decoder obtains the code distribution $\Omega^{\tau,s}$ of Ψ for which the number of doped symbols is smallest, a seed for the matrix construction of $G_{\tau,s}$ and the use of $\Omega^{\tau,s}$, the number of segments $|\Gamma_\tau|$ and corresponding LLR values send by the encoder. Using $(y_{\tau,(N+1)}, \ldots, y_{\tau,(N+m_{\tau,s})})$, the decompression is applied to the level τ sequence $(b_\tau(y_1), \ldots, b_\tau(y_N))$, all the sequences at lower levels 1 though $\tau - 1$ have been already recovered and the conditional LLRs needed by the SP decoder at level τ are known:

a) From the row vector $(y_{\tau,(n+1)}, \ldots, y_{\tau,(N+m_{\tau,s})})$, reconstruct the intermediate bits $(z_{\tau,1}, \ldots, z_{\tau,N})$ using identical iterations of the combined SP-CLID algorithm performed at the FC part.

b) Calculate the bits $(b_\tau(y_1 1), \ldots, b_\tau(y_N))$ using the matrix $G_{\tau,s}$ (inverse matrix of $G_{\tau,s}^{-1}$) [7, 6]:

$$(b_\tau(y_1), \ldots, b_\tau(y_N)) = G_{\tau,s} \cdot (z_{\tau,1}, \ldots, z_{\tau,N}).$$

2. From the row vector $(y_1, \ldots, y_N)$, apply the inverse of transformation of the GST to recover the original RSDS.

3. Separately, an inverse of the EC recovers the original RLDS.

4. Finally, apply an inverse BWT to recover the original text sequence using the sequence reconstructed from the RSDS and the RLDS.

4.1.6 Numerical Results

We evaluated the compression rates of our scheme with the Calgary Corpus and large Canterbury Corpus files and compared it to leading text compression system *gzip-b*, *bzip-9* and *ppmd5* (see [1]). The results[6] are summarized below; *ifount06* is the approach presented in this chapter, with the GST stage being the IFC.
As can be seen in tables Tab. 4.3 and Tab. 4.4, the scheme proposed is comparable to schemes presented in the literature, for large files. However, our scheme is better and more robust for joint source channel coding for transmission over noisy channels.

Compression rates for the Calgary Corpus					
File	Size	ifount06	gzip-b	bzip-9	ppmd5
bib	111261	2.39	2.51	1.95	1.89
book1	768771	2.62	3.25	2.40	2.34
book2	610856	3.31	2.70	2.04	1.98
geo	102400	4.96	5.34	4.48	4.96
news	377109	2.77	3.06	2.51	2.42
obj2	246814	2.88	2.63	2.46	2.35
Average	369536	3.15	3.24	2.64	2.65

Table 4.3: Calgary results

[6] A text compression rate is measured by dividing the number of bits of the compressed text by the number of bytes of the uncompressed text.

Compression rates for the Large Canterbury Corpus					
File	Size	ifount06	gzip-b	bzip-9	ppmd5
E.coli	4638690	1.89	2.24	2.13	1.99
bible.txt	4047392	1.62	2.33	1.65	1.58
world192.txt	2473400	1.71	2.33	1.57	1.52
Average	3719828	1.74	2.30	1.79	1.70

Table 4.4: Canterbury results

4.2 Distributed Source Coding

It is well known that a rate $R = H(X, Y)$ suffices to encode the sources X and Y when the encoder compress these sources together. J. Wolf and D. Slepian have demonstrated [49] that by even separately compressing X and Y, the rate R is still sufficient to ensure lossless recovery of sources X and Y at the decoder side. The Slepian-Wolf theorem has been proved using the random binning strategy [9], which is not useful for practical applications such as video or text source coding.

A. Liveris et al. [31] considered the compression of a memoryless source with side information using the syndrome-LDPC based approach. The problem of designing good practical source codes for correlated sources using rateless codes has been investigated in [15, 50]. In [15], the authors optimized a rate-adaptive source coding-they called Matrioshka codes-with side information, which sends additional syndromes in layers in order to force the convergence of the decoding, extending the approach considered in [31]. In [50], the authors introduced another rate-adaptive scheme unifying LDPC and accumulate codes.

4.2.1 Scheme using Multilevel based SP Decoding

In this section, we consider the symmetric distributed source coding using Fountain codes (Fig. 4.2). The objective is to recover the source X, and the source Y based on the knowledge of X. The scheme proposed is strongly re-

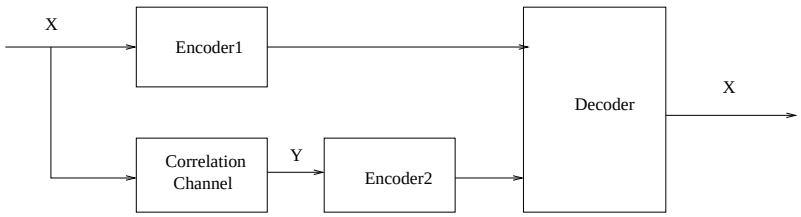


Figure 4.2: Symmetric distributed source coding scenario

lated to the Fountain-code-based single-source coding introduced in [7]. However, contrary to [7], the encoder could not necessarily verify before transmitting whether the decoder could successfully decode. The proposed scheme

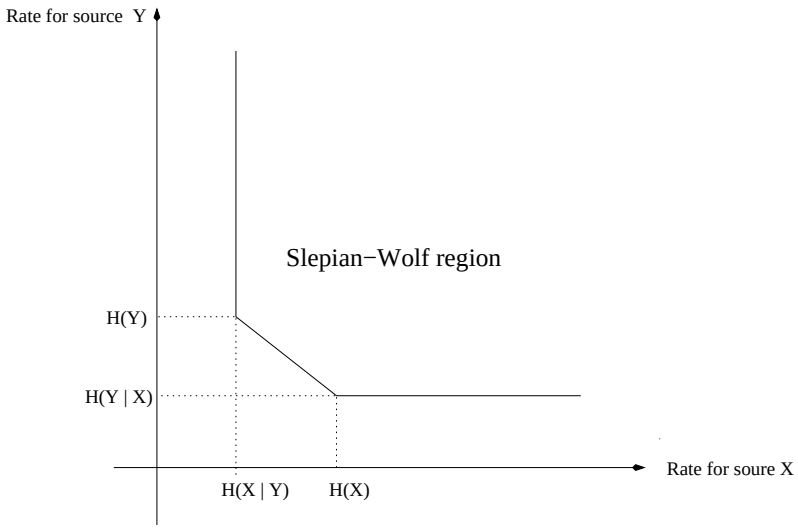


Figure 4.3: Graphical Slepian-Wolf region

follows a simple strategy called the Blind Iterative Doping (BID) together with the SP decoding algorithm.

PROBLEM SET-UP

We assume that both encoders and the joint decoder know in advance the LT probability distribution Ω during the transmission steps. Assumptions for the the sources $(X) = (x_1, \cdots, x_N), (Y) = (y_1, \cdots, y_N)$ are given as follows.

- The sources X and Y represent binary sources observed at different encoders, where the x_i's and y_i's are independent and *not necessarily* identically distributed, where $Pr(x_i) = 1 - Pr(y_i)$, where $i \in \mathcal{I}_1^N$, $x_i, y_i \in \{0, 1\}$. For the pure Slepian-Wolf scenario, sources X and Y are separately encoded and jointly decoded, with rates R_X, for source X (respectively, R_Y, for source Y) such that

$$R_X \geq H(X|Y), R_Y \geq H(Y|X), R_X + R_Y \geq H(Y, X) \tag{4.1}$$

where H denotes the joint entropy of (X, Y).
If the source X is encoded on a perfect noiseless channel and the source Y is source-channel encoded on a a given noisy channel with capacity C (in bits per channel use), the relation (1) can be written

$$R_X \geq H(X|Y), C \geq H(Y|X) R_Z, R_X + R_Y \geq H(Y, X) \tag{4.2}$$

where $Z = (z_1, \cdots, z_M)$ is the source-channel encoder output, the input being Y, and $R_Z = \frac{N}{M}$ is the corresponding source-channel rate.

- Let $[(x_1, y_1), \cdots, (x_n, y_n)]$ be a sequence of jointly distributed i.i.d. variables. The correlation between sources are not known at the two encoders. The two sources X and Y are encoded independently from each other. For our experiments, we assume that the correlation between sources is generated by considering the BSC channel correlation with

$$Pr(x_i = y_i) = 1 - p, \text{ where } 0 < p < 0.5. \tag{4.3}$$

  We assume that the parameter p is unknown by encoders [7] and the joint decoder.

APPROACH

In this section, we describe how encoders and the joint decoding work, for a given LT code with parameters (N, Ω).

Encoding: We compress X using the algorithm described in [31]. We use the following strategy to compress Y, without any knowledge of X:

- First, we calculate a vector of binary intermediate bits $I = (i_1, \cdots, i_N)$, from the binary symbols $Y = (y_1, \cdots, y_N)$ through a linear invertible $N \times N$ matrix G [7, 6]:

$$I = G^{-1} \cdot Y$$

  where G is obtained using Algo. 2 in Chapter 1.

- Apply the LT encoding rule to the sequence I (using Algo. 1 in Chapter 1). We thus obtain a sequence of M bits $S = (s_1, \cdots, s_M)$.

- Next, we generate incrementally $C = (c_1, \cdots, c_M)$ such that $c_1 = s_1, c_2 = s_1 + s_2, \cdots, c_M = s_{M-1} + s_M$.

- A bipartite graph BG (see Fig. 4.4) is set up between nodes corresponding to I and those corresponding to Y, and between nodes corresponding to I and those corresponding to C. The row vector of bits C constitutes the payload of the compressed bits, which is sent to the decoder.

Decoding: The goal of the decoder is to recover losslessly the N-length sequences X and Y. To decode X, we use the decoding strategy described in [31]. To decode the sequence Y, the decoder proceeds as follows:

- Construct the bipartite graph BG.

- The SP decoding algorithm is applied to BG as follows. The objective of the SP algorithm is to decode the symbols I using the full knowledge of the symbols C and the LLR of Y based on the knowledge of C.
  Initial LLRs of Y are $\log(\frac{Pr(x_j=0|y_j)}{Pr(x_j=1|y_j)} = (1 - 2y_j)\log(\frac{1-p_{est}}{p_{est}}))$, those of

[7] This is only for our experiments. In general, one can assume that $P(y_i = 1|x_i)$ is unknown by encoders and the joint decoder

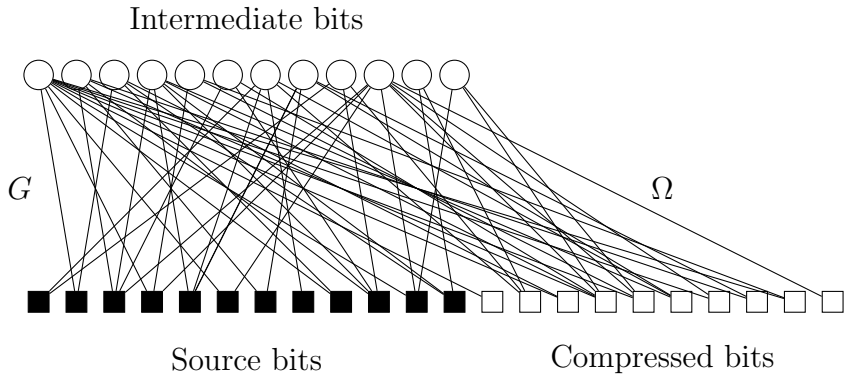


Figure 4.4: Tanner graph for the Two-layer LT approach

C are $+\infty$ (if $c_k = 0$) or$-\infty$(if $c_k = 1$) and those of I are 0, where $j \in \mathcal{I}_1^N, k \in \mathcal{I}_1^M$ and p_{est} is an estimated cross-over probability. The way, the estimation is done is described below.

- During the SP-algorithm, a strategy, called Blind Iterative Doping (BID) is applied.
  During the BID, the encoder randomly picks a bounded number of bits among the bits $i_1, \cdots, i_N$ and sends them to the decoder to help the SP decoding to converge. It's assumed that, every f rounds, the encoder sends g bits to the decoder, where $f > 0$ and $g > 0$ are design parameters. These g randomly chosen bits are called pseudo-doped bits.
  Recall that the parameter p is unknown by encoders and the joint decoder. The encoder starts transmitting $M_1 = NH(p_{e_1})$ symbols $c_1, \cdots, c_{nH(p_{e_1})}$ and pseudo-doped bits until the decoder has enough information to decode, at which time an acknowledgement message is sent to the encoder, where p_{e_1} is a first estimated value of p. As soon as the decoder has enough information, the encoder moves from $p_{e_j} > p_{e_{j+1}}$, sending $n(H(p_{e_{j+1}}) - H(p_{e_j}))$ symbols in addition to the $nH(p_{e_j})$ symbols previously sent to the decoder, where $p_{e_j} = H^{-1}(\frac{M_j}{N})$ and $j > 0$. This process is repeated v times, where $p_{e_v} > \cdots > p_{e_1}$, $p_{e_v} \approx p$ and $v > 0$. It is assumed that Encoder2 sends a seed to the decoder, allowing him to locate pseudo-doped symbols that are chosen and the reconstruction of the matrix G.

NUMERICAL RESULTS

Simulations results of the LT-BLID approach are evaluated and compared with those of [50]. The codeword length is $n = 396$ for BSC(p) statistics between X and Y, where X is Bernoulli(0.5)[8] source and $0.2 < H(Y|X) < 0.7$.

[8] X is Bernoulli(0.5), which means that the source X is available at the decoder and acts as a side information for the decoding of Y

The proposed LT-BLID approach performs around within 10 per cent of the

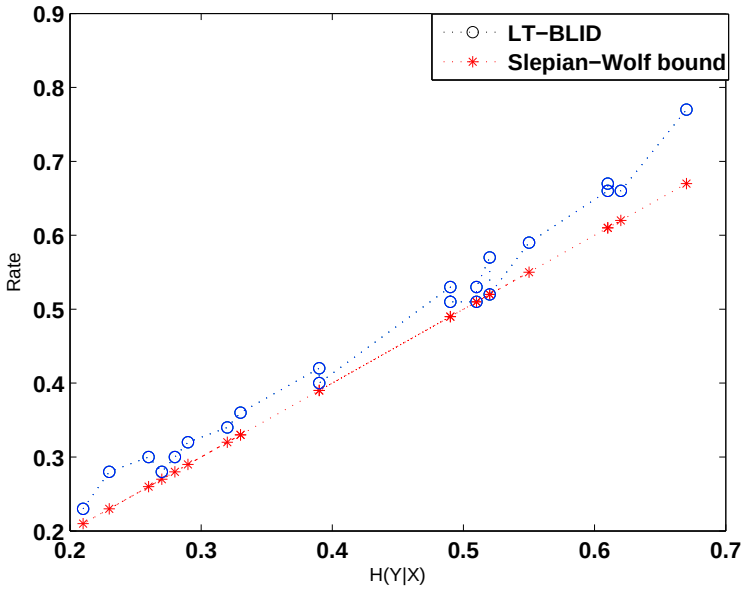


Figure 4.5: Performance of LT-BLID codes of length 396 bits over BSC

Slepian-Wolf bound which is as good as [50].

4.3 Conclusions

In this chapter, we provided a compression algorithm using a multilevel approach based on systematic LT codes. Our algorithm was based on the Burrows-Wheeler Transform. The proposed scheme follows the Closed-Loop Iterative Doping algorithm together with the multilevel stage decoding Sum-product at the Frequency Count stage. Our algorithm offered encouraging compression rate performance for large files. We also described one solution to the two-user Slepian-Wolf problem in a certain part of the achievable region using fountain codes. Symmetric case of memoryless compression of two correlated sources was considered and modeled by binary symmetric channels. The compression was done by two separate compressors without any exchange of information between them. The decompressor used SP decoding algorithm in conjunction with the Blind Iterative Doping strategy. Simulation results indicate performance close to the Slepian-Wolf limit.

Bibliography

[1] The Canterbury Corpus link: http://corpus.canterbury.ac.nz.

[2] J. Abel and W. Teahan. Universal Text Preprocessing for Data Compression. *IEEE Transactions on Computers*, pages pages 497–507, 2005.

[3] A.W.Eckford, F.R.Kschischang, and S.Pasupathy. Analysis of Low-Density parity-Check Codes for the Gilbert-Elliott Channel. *IEEE Trans. Inform. Theory*, 51:3872–3889, 2005.

[4] J. Byers, M. G. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of* ACM SIGCOMM*, Vancouver, BC, Canada*, pages 56–67, 1998.

[5] Haixiao Cai, Kulkarni S.R., and Verdu S. Universal entropy estimation via block sorting. *IEEE Transactions on Information Theory*, 50:1551 – 1561, July 2004.

[6] G. Caire, S. Shamai A, A. Shokrollahi, and S. verdu. Fountain Codes for lossless data compression. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, A. Barg and A. Ashkhimin, Eds. American MathematicalSociety*, 2005.

[7] G. Caire, S. Shamai A., A. Shokrollahi, and S. verdu. Universal variable-length data compression of binary sources using Fountain Codes. *Information Theory Workshop, 2004. IEEE*, pages 123–128, 29 Oct. 2004.

[8] G. Caire, S. Shamai, and S. Verdu. A new data compression algorithm for sources with memory based on error correcting codes. pages pages 291–295, 2003.

[9] Thomas M. Cover. A proof of the data compression theorem of Slepian and Wolf for ergodic sources. *IEEE Trans. Inform. Theory*, 21:pages 226–228, jul. 1975.

[10] Baron D. and Bresler Y. An O(N) semipredictive universal encoder via the BWT. *IEEE Transactions on Information Theory*, 50:928 – 937, May 2004.

[11] M. Davey and D. MacKay. Low Density parity Check Codes over GF(q). *IEEE Commun. Lett.*, 2(1):165–167, June 1998.

[12] D. Declerq and M. Fossorier. Decoding algorithms for nonbinary LDPC codes over GF(q). *IEEE Transactions on Communications*, 55:633–643, 2007.

[13] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977.

[14] S. Deorowicz. Universal lossless data compression algorithms. *Doctoral dissertation, Silesian University of Technology, Gliwice, Poland*, 2003.

[15] A. Eckford and W. Yu. 'Rateless Slepian-Wolf codes. In *Proc. Asilomar Conference on Signals, Systems and Computers*, 2005.

[16] E. O. Elliott. Estimates of error rates for codes on burst-noise channels. *Bell System Technical Journal*, 42:1977–1997, Sep. 1963.

[17] O. Etesami and M. A. Shokrollahi. Raptor codes on binary memoryless symmetric channels. *IEEE Transactions on Information Theory*, 52:2033–2051, 2006.

[18] R. G. Gallager. *Low Density Parity Check Codes*. MIT Press, 1963.

[19] A. J. Goldsmith and P. P. Varaiya. Capacity, mutual information, and coding for finite-state Markov channels. *IEEE Trans. Inform. Theory*, 42(3):868–886, May 1996.

[20] Itoh H. and Tanaka H. An Efficient Method for in Memory Construction of Suffix Arrays. *IEEE String Processing and Information Retrieval Symposium (SPIRE'99)*, pages 81– 88, september 1999.

[21] Kao T. H. Improving Suffix-Array Construction Algorithms with Applications. *Master's thesis, Gunma University, Kiryu, Japan*, pages 376–8515, 2001.

[22] Abel J. A fast and efficient post BWT-stage for the Burrows-Wheeler Compression Algorithm. *Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, J. A. Storer and M. Cohn, Eds.*, 449, 2005.

[23] Karkkainen J. and Sanders P. Simple Linear Work Suffix Array Construction. *30th International Colloquium on Automata, Languages and Programming, number 2719 in LNCS, Springer*, pages 943–955, 2003.

[24] Larsson N. J. and Sadakane K. Faster Suffix Sorting. *Technical report 1999*, 1999.

[25] Sadakane K. Unifying Text Search and Compression - Suffix Sorting, Block Sorting and Suffix Arrays. *Ph.D. Dissertation, Department of Information Science, Faculty of Science, University of Tokyo*, 2000.

[26] Visweswariah K., Kulkarni S, and Verdu S. Output distribution of the Burrows-Wheeler transform. *Information Theory, 2000. Proceedings. IEEE International Symposium on 25-30 June 2000 Page(s):53.*

[27] R. Karp, M. Luby, and A. Shokrollahi. Verification decoding of Raptor codes. In *Proc. of ISIT 2005*, pages 1310–1314, 2005.

[28] R. E. Krichevsky and V. K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27:199–207, March 1981.

[29] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[30] X. Li and M.R.Soleymani. A proof of the Hadamard Transform Decoding of the Belief Propagation Decoding for LDPCC over GF(q). *Vehic. Tech. Conf.*, Sept. 2004.

[31] A. Liveris, Z. Xiong, and C. Georghiades. Compression of binary sources with side information at the decoder using LDPC codes. *IEEE Commun. Lett.*, Lett. 6(10):pages 440–442, 2002.

[32] M. Luby and M. Mitzenmacher. Verification codes. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, 2002.

[33] M. Luby, M. Mitzenmacher, and A. Shokrollahi. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. Inform. Theory*, 47:585–598, 2001.

[34] M. G. Luby. LT codes. In *Proceedings of 43rd IEEE Symposium Foundations of Computer Science*, pages 1–10, 2002.

[35] Burrows M and Wheeler D. J. A Block-Sorting Lossless Data Compression Algorithm. *Technical report, Digital Equipment Corporation, Palo Alto, California, 1994.*

[36] D. MacKay and M. Davey. Evaluation of Gallager codes for short block length and high rate applications. In *In Codes, Systems and Graphical Models*, pages 113–130. Springer-Verlag, 2000.

[37] S. Mohajer and A. Shokrollahi. Raptor Codes with Fast Hard Decision Decoding Algorithms. In *Information Theory Workshop, 2006. ITW '06 Chengdu. IEEE*, pages 56–60, 2006.

[38] M. Mushkin and I. Bar-David. Capacity and coding for the Gilbert-Elliott channels. *IEEE Trans. Inform. Theory*, 35(6):1277–1290, Nov. 1989.

[39] T. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47:599–618, 2001.

[40] Deorowicz S. Second step algorithms in the Burrows-Wheeler compression algorithm. *Software - Practice and Experience*, 2002.

[41] David Salomon. Coding for DATA AND COMPUTER COMMUNICATIONS. *Springer Science*, pages pages 68–110, 2005.

[42] Khalid Sayood. Lossless Compression Handbook, Academic Press. 2003.

[43] G. I. Shamir. *Universal coding for classes of nonstationary sources Lower bounds and optimal schemes*. Ph.D. Thesis, University of Notre Dame, 2000.

[44] G. I. Shamir and D. J. Costello. Asymptotically optimal low-complexity sequential lossless coding for piecewise-stationary memoryless sources-part I: the regular case. *IEEE Transactions on Information Theory*, 46(7):2444–2467, Nov. 2000.

[45] G. I. Shamir and N. Merhav. Low-complexity sequential lossless coding for piecewise-stationary memoryless sources. *IEEE Transactions on Information Theory*, 45(5):1498–1519, Jul. 1999.

[46] G. I. Shamir, T. J. Tjalkens, and F. M. J. Willems. Universal noiseless compression for noisy data. In *Information Theory and Applications Workshop, San Diego, California*, 2007.

[47] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52:2551–2567, 2006.

[48] A. Shokrollahi and W. Wang. LDPC codes with rates very close to the capacity of the q-ary symmetric channel for large q. In *Proceedings of the International Symposium on Information Theory, Chicago*, 2004.

[49] D. Sleepian and J.K. Wolf. Noiseless coding of correlated information sources. *IEEE Trans. Inform. Theory*, 19:pages 471–480, jul. 1973.

[50] D. Varodayan, A. Aaron, and B. Girod. Rate-adaptive codes for distributed source coding. In *Signal Processing: Special Issue on Distributed Source Coding*.

[51] W. Zhang, D. J. Costello, T. E. Fuja, G. I. Shamir, and A. W. Eckford. Iterative estimation and decoding for Gaussian channels with abruptly changing statistics. In *Proc. IEEE International Symposium on Information Theory,* Seattle, WA, USA, pages 2466–2470, 2006.

[52] W. Zhang, C. Koller, A. W. Eckford, D. J. Costello, T. E. Fuja, and G. I. Shamir. Estimation and decoding strategies for channels with abruptly changing statistics. In *Proc. IEEE Information Theory Workshop,* Rotorua, New Zealand, 2005.

Curriculum Vitae
Bertrand NDZANA NDZANA

Education:

- **École Polytechnique Fédérale de Lausanne (EPFL)**
  Ph.D. in computer and communication sciences
  Rresearch topic: Source and Channel problems using Fountain codes
  2004 - 2009 (expected)

- **EPFL**
  Swiss Federal diploma of engineering
  Communication systems division
  1998 - 2004

Professional Experience:

- **École Polytechnique Féd'erale de Lausanne(EPFL)**
  Research assistant; research in small teams in applied mathematics,teaching, supervision of semester projects
  2004 - 2009

- **University of York, Toronto, Canada**
  3 Internships of 6 weeks; Channel Coding
  2006-2007-2008

- **Orange Communications, Lausanne,Switzerland**
  Master thesis of 6 months; Research and developement in security/GPRS attacks
  2004

- **IDQuantique, Geneva, Switzerland**
  Internship of 6 months; Quantic cryptography/BB84 protocol implementation

2003

- **Motorola, Geneva,Switzerland**
  Internship of 3 weeks; Classic cryptography/Software developement
  2001

- **CREM, Martigny, Switzerland**
  Internship of 3 months, web-programmer
  1999

Conference Publications:

1. H. Cronie, B. Ndzana Ndzana, A. Shokrollahi, "*Decoding Algorithms for Binary Raptor Codes over Nonbinary Chanels*", Proceedings IEEE International Symposium on Information Theory, Seoul (Korea), June 28 - July 3, 2009

2. B. Ndzana Ndzana, A. Eckford, A. Shokrollahi, G. I. Shamir "*Fountain Codes for Piecewise Stationary Channels*", Proceedings IEEE International Symposium on Information Theory, Toronto, Ontario (Canada), July 6-11, 2008

3. B. Ndzana Ndzana, A. Shokrollahi, "*Fountain Codes for the Slepian-Wolf problem*", Proceedings of Forty-Fourth Annual Allerton Conference on Communication, Control, and Computing, Illinois (USA), September 27-29, 2006

4. B. Ndzana Ndzana, A. Shokrollahi, J. Abel, "*Burrows-Wheeler Text Compression with Fountain Codes*", Proceedings of Data Compression Conference, Snowbird (USA), March 28-30, 2006