Data Compression using Error Correcting Codes

Javad Haghighat

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

March 2007

Your file Votre référence
ISBN: 978-0-494-30135-7
Our file Notre référence
ISBN: 978-0-494-30135-7

ABSTRACT

Application of error correcting codes for data compression is first investigated by Shannon where he suggests that there is a duality between source coding and channel coding. This duality implies that good channel codes are likely to be good source codes (and vice versa). Recently the problem of source coding using channel codes is receiving increasing attention. The main application of this problem is when data are transmitted over noisy channels. Since standard data compression techniques are not designed for error correction, compressing data and transmitting over noisy channels may cause corruption of the whole compressed sequence. However, instead of employing standard compression techniques, like Huffman coding, one may compress data using error correcting codes that are suitable for both data compression and error correction purposes. Recently, turbo codes, repeat-accumulate codes, low density parity check codes, and fountain codes have been used as lossless source codes and have achieved compression rates very close to the source entropy. When a near-lossless compression is desired, i.e. a small level of distortion is acceptable, the source encoder generates fixed-length codewords and the encoding complexity is low. Theoretically, random codes could achieve near-lossless compression. In literature, this has been proved by presenting a random binning scheme. Practically, all powerful channel codes, e.g. turbo codes, can follow the same procedure as suggested in random binning and achieve compression rates close to the entropy. On the other hand, if a completely lossless compression is required, i.e. if the distortion must be forced to zero, the source encoding is a complicated iterative procedure that generates variable-length codewords to guarantee zero distortion. However, the large complexity of encoding imposes a large delay to the system. The iterative encoding procedure can be regarded as using a nested code where each codeword of a higher-rate code is formed by adding parities to a codeword of some lower-rate code. This iterative encoding is proposed for practical codes, e.g. turbo codes and low density parity check (LDPC) codes,

in the literature. In contrast to near-lossless source coding, in the lossless case no random coding theory is available to support achievability of entropy and specify distribution of the compression rate.

We have two main contributions in this thesis. Our first contribution is presenting a tree structured random binning scheme to prove that nested random codes asymptotically achieve the entropy. We derive the probability mass function of the compression rate and show how it varies when increasing the block length. We also consider a more practical tree structured random binning scheme, where parities are generated independently and randomly, but they are biased. Our second contribution is to decrease the delay in turbo source coding. We consider turbo codes for data compression and observe that existing schemes achieve low compression rates; but because of large block length and large number of iterations they impose a large delay to the system. To decrease this delay we look at the problem of source coding using short block length turbo codes. We show how to modify different components of the encoder to achieve low compression rates. Specifically we modify the parity interleaver and use rectangular puncturing arrays. We also replace a single turbo code by a library of turbo codes to further decrease the compression rate. Since the scheme is variable-length and also many codes are used, the codeword length along with the code index (index of the turbo code which is used for compression) are transmitted as an overhead. Transmission of this overhead increases the compression rate. We propose a detection method to detect this overhead from the codeword. Therefore, the overhead is no longer transmitted since it is detected from the codeword at the decoder. This detection method will reduce the compression rate for short block length systems but it becomes less attractive for large block length codes.

ACKNOWLEDGEMENTS

First of all I would like to express my deepest gratitude to my supervisors Prof. M. Reza Soleymani and Prof. Walaa Hamouda, for their continual support and guidance during my studies at Concordia University. Also, the financial support provided by them is gratefully acknowledged.

I would like to thank the committee members of my Ph.D. defense: Prof. Fady Alajaji, Prof. Ahmed K. Elhakeem, Prof. Rajamohan Ganesan, Prof. Yousef R. Shayan, and the committee chair Prof. Oscar A. Pekau, for their feedback and suggestions that improved the quality of this final version. My special thanks goes to the external examiner, Prof. Fady Alajaji of Queen's University, for his technical insight and the suggestions he made for the future research. I would like to thank Prof. Eugene Plotkin for his thoughtful criticism that improved the quality of my work.

This research was conducted in the Wireless and Satellite Communications Lab of Concordia University. I would like to thank all my friends in the Wireless and Satellite Communications Lab for their collaboration and support. Specially, I would like to thank my dear friend Hamid Behroozi for his great support and for many fruitful discussions we had during this work. I would like to thank all my friends and relatives for motivating me and believing in me during all these years.

The last but not the least, I appreciate the love and support I have always received from my family. I had the privilege to grow up in a family that valued the importance of education and always encouraged me during my studies. I will never forget my late brother Ardavan who loved mathematics and taught me my first courses in mathematics. I also thank my brother Afshin who has supported me all over my life. But more than anyone else, I owe this to my mother, Sedigheh, and my father, Parviz, for their unconditional love and for all the sacrifices they made for me.

This Thesis is dedicated to my father, Parviz, my mother, Sedigheh, my brother, Afshin, and my late brother, Ardavan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

bps	bits per sample
BCJR algorithm	Bahl-Cocke-Jelinek-Raviv algorithm
BER	Bit Error Rate
cmf	cumulative mass function
i.i.d.	independent identically distributed
LDPC code	Low Density Parity Check Code
pmf	probability mass function
RA code	repeat-accumulate code

List of Symbols

X	Information source
χ	Source alphabet
$\mathbf{x}$	Message sequence
$\widehat{\mathbf{x}}$	Reconstructed message sequence
$\mathbf{y}$	Codeword
$d(\mathbf{x}, \widehat{\mathbf{x}})$	Distortion between two sequences
D	Expected value of distortion
n	Message block length
l	Codeword length
$\overline{l}$	Expected value of codeword length
R	Compression rate
$\overline{R}$	Expected value of compression rate
R_i	Compression rate achieved by ith code in a library of codes
$\overline{R}_b$	Expected value of compression rate for tree structured random binning
$\widetilde{R}_b$	Expected value of compression rate for a completely lossless scheme
A_ε	ε-typical set
$\lvert A_\varepsilon \rvert$	Number of typical sequences
N_0	Number of typical sequences minus one
$g(\varepsilon)$	Probability of observing typical sequence
p	Probability of observing one in source bit stream

m	Number of parities added in each stage of tree structured random binning
M	Number of codes in a library of codes
N_h	Number of rows in a puncturing array
N_v	Number of columns in a puncturing array
q_m	Probability mass function of tree structured random binning
$q_m(l)$	Probability of observing codewords with length l in tree structured random binning
$\mathbf{L}$	L-value representing source *a priori* knowledge
$\mathbf{A}$	*a priori* L-value
$\mathbf{E}$	Extrinsic l-value
P_d	Probability of detection error
$\overline{R}_d$	Expected value of compression rate by using code detection method
$\overline{l}_d$	Expected value of codeword length by using code detection method

Chapter 1

Introduction

We consider the problem of data compression using error correcting codes. In this chapter we first explain the motivation behind the problem. Then we briefly introduce previous works on data compression using error correcting codes. Section 3 summarizes the main contributions of this thesis. Finally, an overview of the remaining chapters is presented.

1.1 Motivation

Source coding and channel coding are dual problems. Source coding (data compression) attempts to remove the redundancy existing in the message, whereas channel coding adds redundancy to the message (in a controlled fashion) in order to combat errors occurring in a noisy channel. The area of channel coding has achieved a state of maturity where powerful error correcting codes have been designed that can approach the capacity of different communication channels. For example, for additive white Gaussian noise (AWGN) channel, the capacity is practically reached by designing appropriate low density parity check (LDPC) or turbo codes. The duality of source coding and channel coding motivates the application of successful channel coding techniques to the dual problem of data compression. Data compression using error correcting codes is especially suitable for cases

where data are transmitted over noisy channels. Since standard data compression techniques are not designed for error correction, compressing data and transmitting over noisy channels may cause corruption of the whole compressed sequence. However, instead of employing standard compression techniques, like Huffman coding, one may compress data using error correcting codes that are suitable for both data compression and error correction purposes. Recently, turbo codes, repeat-accumulate codes, low density parity check codes, and fountain codes have been used as lossless source codes and have achieved compression rates very close to the source entropy.

In this thesis, we mainly consider fixed-to-variable length data compression using error correcting codes. In this problem, the encoder maps fixed length message blocks to variable length codewords. Compared to standard data compression techniques, like Lempel-Ziv coding [1], designed source codes achieve lower compression rates. Another attractive feature of these fixed-to-variable length schemes is their ability to trade off data compression for error correction. In the presence of noise, one can remove less redundancy and allow better error correction capability. It will be shown that this could be done in a very natural way. The compressed sequence consists of parity bits of a channel code. Thus, removing less parity bits is equivalent to applying a more powerful channel code.

1.2 Data Compression Using Error Correcting Codes

The application of error correcting codes to data compression is inspired by Shannon's observation on duality of source coding and channel coding. In his landmark paper [2] Shannon remarks:

"There is a curious and provocative duality between the properties of a source with a distortion measure and those of a channel. This duality is enhanced if we consider channels in which there is a cost associated with the different input letters ... It can be shown readily that this [capacity cost] function is concave downward. Solving this

problem corresponds, in a sense, to finding a source that is right for the channel and the desired cost ... In a somewhat dual way, evaluating the rate distortion function for a source ... the solution leads to a function which is convex downward. Solving this problem corresponds to finding a channel that is just right for the source and allowed distortion level."

Shannon's work has been followed by others to construct a solid theory on duality of source and channel coding. In [3] the duality of channel coding and source coding has been interpreted as a sphere packing versus a sphere covering problem. It is shown that for Gaussian sources, channel coding is a sphere packing problem whereas source coding is a sphere covering problem. This implies that a good channel code is likely to be a good source code [22]. In [5] the duality between channel capacity and rate-distortion function has been extended to the case where side information is available at both encoder and decoder sides. As another work that studies the duality between source and channel coding we can refer to [6]. The authors in [6] consider the extension of this duality to the side information case where two correlated sources are compressed together. As practical implementations of this duality we can mention trellis-coded quantization [7], [8] in source coding which has been inspired by trellis-coded modulation in channel coding. As another example of this duality we may mention the shaping of constellation points [9] in channel coding which has been inspired by constraining the entropy of quantization points [10] in source coding.

Perhaps the most attractive application of the duality between source and channel coding is the problem of lossless compression of two correlated sources. Because of the correlation, one source can be considered as the noisy version of the other source after passing through a channel. Since lossless compression is desired, the distortion measure (which usually is the probability of error) must approach zero. This is an exact match for the channel coding case where the probability of error approaches zero. Therefore, a channel code can be designed to achieve the lossless compression. It is interesting to know that even the main proof of achievable rate region for lossless compression of correlated

sources is based on random codes [11]. Based on this observation, many codes including turbo codes, low-density parity check (LDPC) codes and repeat-accumulate codes have been used for lossless compression of correlated sources where all achieved rates are very close to the theoretical limits. Related works on this area can be found in [12]-[21] and references therein.

In contrast to compressing correlated sources, lossless compression of a single source using a channel code has received less attention. However, this problem has also been considered in some references including [23]-[31]. In fact, one can use a random binning method [3] to show that random codes can be used for lossless data compression. In that sense, it is shown that by increasing the data block length, these random codes asymptotically achieve the entropy limit. Therefore, a random-like code (e.g. turbo code or LDPC code) could be employed as a source code. For example, in [27], a turbo code is used to compress a biased binary independent identically distributed (*i.i.d.*) source. The encoder, in [27], generates two parity sequences using two parallel concatenated convolutional codes. Compression is achieved by heavily puncturing these parities.

The random binning argument presented in [3] considers a fixed-to-fixed length source coding scheme, i.e. a fixed-length message block is mapped to a fixed-length codeword. Although fixed-to-fixed length schemes asymptotically achieve the entropy, fixed-to-variable (or variable-to-fixed) length codes achieve lower compression rates in finite block length systems. To design a fixed-to-variable length code, one may use a library of parallel random binning schemes with different rates. As a practical implementation, [26] considers lossless compression of a biased *i.i.d.* binary source using a Hamming code, a BCH code and a rate-one code. As an alternative to using a library of codes, a fixed-to-variable length source code can be implemented by a nested code, where each codeword of a higher rate code is generated by adding bits to a codeword of a lower rate code. For example, fixed-to-variable length LDPC and turbo source coders have been designed in [29] and [22] respectively (alternative schemes based on fountain and repeat-accumulate (RA) codes are presented in [30] and [31] respectively). In [22],

after encoding the message by a turbo code, the parities are gradually punctured using an iterative algorithm. Puncturing continues as long as the codeword is decodable with no errors. However, in [28], after transmitting the syndrome of the message, the message bits are sent one by one using an iterative doping approach until the transmitted bits are sufficient to decode the entire message at the decoder. If we refer to the method of [22] as a decremental redundancy method, the method in [28] can be referred to as an incremental redundancy method.

1.3 Thesis Contributions

Contributions presented in this thesis can be summarized as follows:

- We consider the theoretical performance of nested error correcting codes for data compression. For this, we generalize the conventional random binning argument to a so called *tree structured random binning* concept. We derive the distribution of the compression rate achieved by the proposed tree structured random binning scheme. Comparing this distribution with the distribution achieved using a library of random binning schemes, we prove that a nested code can achieve rates close to a library of codes but with much lower encoding/decoding complexity.

- For large data blocks, the proposed scheme is shown to asymptotically achieve the entropy limit.

- As a practical implementation of this random binning scheme, we examine the performance of the lossless turbo source coding scheme proposed in [22]. Considering short block length codes, we modify different components of the encoder to achieve better compression rates.

- We present a detection algorithm to eliminate the overhead introduced by the transmission of the codeword length and the number of codes. Using such an iterative

detection technique proves to further reduce the compression rate for short block length turbo source codes.

When we establish the tree structured random binning theory, we generally speak about typical and atypical sequences [3], i.e. no specific source is considered in this case and the theory is applicable to all types of information sources. However, when we discuss turbo source coding, a binary biased independent identically distributed source is considered. This assumption could be justified by noticing the fact that in some cases the source is in fact discrete (like transmission of text) and could be described by a binary sequence without any loss of information. In case of continuous sources, a quantizer is used and thus some amount of information is lost. In this case the considered lossless approach serves as a background for the lossy compression system. The independence is also justified by the use of a whitening filter.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 is devoted to background and literature review. The chapter begins by reviewing the fundamental concepts of data compression. We introduce the random binning argument and fixed-to-fixed length source coding using turbo codes. Then the lossless turbo source coding scheme is presented which is a fixed-to-variable length source coding scheme.

In Chapter 3 we consider data compression using a library of random binning codes and derive the distribution of compression rate (in this case the scheme is fixed-to-variable length thus the compression rate is a random variable). The tree structured random binning theory is established and the distribution of compression rate is derived. Tree structured random binning considers the theoretical performance of a nested channel code applied for data compression purpose. At the end of this chapter we consider the limitations caused by using a practical code instead of a random code for tree structured random

binning.

Chapter 4 begins by noticing the drawback of lossless turbo source coding schemes, i.e. large latency (or system delay). To reduce this delay we propose short block length turbo source codes and modify different components of the encoder to achieve promising compression rates using these short block length codes.

In Chapter 5 we propose a detection method to detect the overhead introduced by the codeword length and the index of applied code. Due to this detection method, the overhead is no longer transmitted since it could be detected at the decoder. Our proposed method helps to reduce the compression rate of short block length systems, but as the block length grows it gradually loses its attraction.

Chapter 6 highlights contributions of this work and gives suggestions for future studies.

The contributions of this thesis are presented in [40]-[47].

Chapter 2

Background

In this chapter we review the concepts of source coding and turbo source coding. Our focus is on lossless source coding, i.e. compressing data without any loss of information. We begin by defining the basic concepts of a source coding system, such as message, block length, and compression rate. Then we talk about typical sequences and define the concept of entropy which serves as a lower bound for the achievable compression rate in lossless source coding. We observe how random codes can achieve this bound by using a random binning method presented in [3]. Then turbo source coding is considered, where we review two original works on data compression using turbo codes. We observe that in [27] turbo codes are used for near-lossless compression, i.e. where a small error probability is allowed. However, in [22] the compression is made completely lossless by checking the decodability of data inside the encoder and prior to transmission. Then we review the numerical results from [27] and [22] to express the achievable compression rates using existing turbo source coding methods.

2.1 Definitions

Consider a discrete, i.i.d source X that takes its outcome x from a finite alphabet χ with cardinality $|\chi|$. We consider a source coding (data compression) system as shown in

Fig. 2.1. For some positive integer n, the source encoder takes a vector of n samples from the source (e.g. $\mathbf{x} = x_1 x_2 ... x_n$) and maps it to an index $\mathbf{y} \in \{1, 2, ..., 2^{nR}\}$, where R is a rational number (between zero and one) such that nR is an integer. Obviously $\mathbf{y}$ could be expressed by nR bits. The source decoder maps $\mathbf{y}$ to a vector $\widehat{\mathbf{x}}$, which is an estimate of $\mathbf{x}$. We further define the parameters as follows:

$\mathbf{x}$ is called the message (sequence).

n is called the block length and is expressed in samples.

R is called the compression rate and is expressed in bits per sample (bps).

$\mathbf{y}$ is called the codeword.

$\widehat{\mathbf{x}}$ is called the reconstructed message (sequence).

The quality of the data compression system is evaluated by a fidelity criterion, or namely a distortion measure. The distortion $d(x, \widehat{x})$ is a (non-negative) measure which specifies the cost of representing the sample x by $\widehat{x}$. Among the most popular distortion measures we can mention:

- Hamming (probability of error) distortion: is given by

$$d(x, \widehat{x}) = \begin{cases} 0 & x = \widehat{x} \\ 1 & x \neq \widehat{x} \end{cases} . \tag{2.1}$$

-Squared error distortion: is given by

$$d(x, \widehat{x}) = (x - \widehat{x})^2. \tag{2.2}$$

The distortion between message sequence $\mathbf{x}$ and reconstructed sequence $\widehat{\mathbf{x}}$ is defined as the average of the per sample distortions, i.e.:

$$d(\mathbf{x}, \widehat{\mathbf{x}}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \widehat{x}_i). \tag{2.3}$$

The total distortion associated with the source code is then defined as:

$$D = E\{d(\mathbf{x}, \widehat{\mathbf{x}})\}, \tag{2.4}$$

where $E\{.\}$ represents the expected value. If the message sequences $\mathbf{x}$ are generated with a probability distribution $p(\mathbf{x})$, then D can be expressed by:

$$D = \sum_{\mathbf{x}} p(\mathbf{x}) d(\mathbf{x}, \widehat{\mathbf{x}}). \tag{2.5}$$

Based on D, we define three types of source coding schemes as follows

- Completely lossless source coding: A source coding scheme is called completely lossless if $D = 0$, i.e. if for any given message sequence $\mathbf{x}$ with $p(\mathbf{x}) > 0$, $d(\mathbf{x}, \widehat{\mathbf{x}}) = 0$.

-Asymptotically lossless (near-lossless) source coding: A source coding scheme is called asymptotically lossless if

$$\lim_{n \to \infty} D = 0. \tag{2.6}$$

-Lossy source coding: A source coding scheme is called lossy, if it is not completely lossless or near-lossless.

In this thesis we mainly deal with completely lossless and near-lossless source coding schemes.

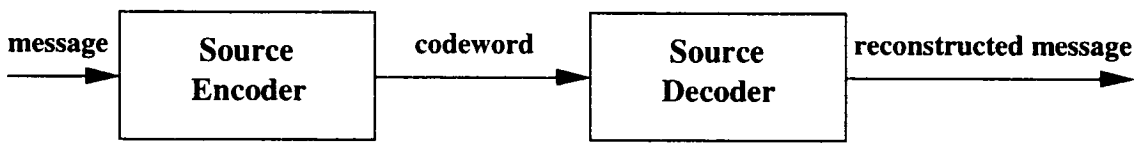

Figure 2.1: The block diagram of a source coding system.

2.2 Typical set and the concept of entropy

As the block length grows, the set of all possible message sequences could be partitioned into two distinct sets. The *typical set* is the set of sequences that are typically observed

as the source outcomes. In other words, typical set is the set of sequences that are typical of the source or represent the type of the source. The *atypical set* (or non-typical set) is the set of all sequences that are not typical. Each sequence belonging to the typical set is called a *typical sequence*. Each sequence belonging to the atypical set is called an *atypical sequence*.

As an example of typical and atypical sequences, let us consider a binary independent identically distributed (i.i.d.) source X, generating binary i.i.d. samples x from an alphabet $\chi = \{0, 1\}$ with $Pr(x = 1) = p$. When the block length, n, is large, we can say that nearly each realization of message $\mathbf{x}$ has np "ones" and $n(1-p)$ "zeros" (let np be an integer). The probability of observing such message $\mathbf{x}$ is $Pr(\mathbf{x}) = p^{np}(1-p)^{n(1-p)}$. Defining a function $H(X) = -plog_2p-(1-p)log_2(1-p)$ gives $Pr(\mathbf{x}) = 2^{-nH(X)}$ (from now on all logarithms used are base 2 otherwise it is mentioned). In other words, for large block length system, the outcome of the source consists of $2^{nH(X)}$ typical sequences each occurring with probability $2^{-nH(X)}$. The function $H(X)$ is called the entropy of the source X. For any given source X with probability distribution $p(x)$, the entropy function is defined as follows [3]:

$$H(X) = -\sum_{\chi} p(x)log(p(x)). \tag{2.7}$$

Similar to the compression rate, the entropy is expressed in bits per sample (bps). Using this definition for entropy, the typical set (or more precisely the ε-typical set) is formally defined as follows [3]:

Definition: For some fixed value $\varepsilon > 0$, the typical set A_ε is the set of message sequences $\mathbf{x}$ with the following property:

$$2^{-n(H(X)+\varepsilon)} \leq p(\mathbf{x}) \leq 2^{-n(H(X)-\varepsilon)}. \tag{2.8}$$

Now we review some important properties of typical set. For this we need to express asymptotic equipartiotion property.

Asymptotic equipartition property [3]: If x_1, x_2, ... are drawn i.i.d. according to probability distribution function $p(x)$, then as $n \rightarrow \infty$, $-\frac{1}{n} log\, p(\mathbf{x}) \rightarrow H(X)$ in probability.

The following theorem is taken from [3] and represents three important properties of the set A_ε

Theorem 2.1- For the set A_ε:

1- $Pr\{A_\varepsilon\} > 1 - \varepsilon$, for n sufficiently large.

2- $|A_\varepsilon| \leq 2^{n(H(X)+\varepsilon)}$, where $|A_\varepsilon|$ denotes the number of elements of A_ε.

3- $|A_\varepsilon| \geq (1-\varepsilon)2^{n(H(X)-\varepsilon)}$, for n sufficiently large.

Proof-

1- From the definition of A_ε in (2.8) we have $Pr\{A_\varepsilon\} = Pr\left\{\left|-\frac{1}{n} log\, p(\mathbf{x}) - H(X)\right| < \varepsilon\right\}$. Then from the asymptotic equipartition property and definition of limit, for any $\delta > 0$, there exists an integer n_1 such that for any $n > n_1$ we have

$$Pr\left\{\left|-\frac{1}{n} log\, p(\mathbf{x}) - H(X)\right| < \varepsilon\right\} > 1 - \delta. \tag{2.9}$$

setting $\delta = \varepsilon$ completes the proof.

2-We have

$$1 = \sum_{\mathbf{x}} p(\mathbf{x})$$

$$\geq \sum_{\mathbf{x} \in A_\varepsilon} p(\mathbf{x})$$

$$\geq \sum_{\mathbf{x} \in A_\varepsilon} 2^{-n(H(X)+\varepsilon)}$$

$$= |A_\varepsilon|\, 2^{-n(H(X)+\varepsilon)}$$

Hence

$$|A_\varepsilon| \leq 2^{n(H(X)+\varepsilon)}. \tag{2.10}$$

3- From part 1 of this theorem $1 - \varepsilon < Pr\{A_\varepsilon\}$, for n sufficiently large. Then similar to the proof for part 2:

$$1 - \varepsilon < Pr\{A_\varepsilon\}$$

$$\leq \sum_{\mathbf{x} \in A_\varepsilon} 2^{-n(H(X)-\varepsilon)}$$

$$= |A_\varepsilon| \, 2^{-n(H(X)-\varepsilon)}$$

Therefore:

$$|A_\varepsilon| \geq (1 - \varepsilon) 2^{n(H(X)-\varepsilon)}.$$

And this completes the proof.

Shannon proves [4] that entropy is the limit of lossless compression, i.e. no lossless source coding scheme could be designed to compress a source X to a rate $R < H(X)$. In other words to achieve lossless compression we must have $R > H(X)$. However, from the concept of typical set we can prove that R can be chosen arbitrarily close to $H(X)$. This could be shown as follows:

Recall that $Pr\{A_\varepsilon\} > 1 - \varepsilon$ (theorem 2.1), for n sufficiently large. Also ε could be chosen arbitrarily small, given large enough n. Thus, the probability of typical set is close to one for sufficiently large block length. This is interesting since the typical set has only about $2^{nH(X)}$ sequences whereas the number of all sequences is 2^n which is much larger (given $(1 - H(X)) >> 0$). This is why the typical set is sometimes referred to as the smallest high-probability set [3]. Using these facts, a lossless source code can be designed as follows:

-Each typical sequences is encoded to $\lceil n(H(X)+\varepsilon)\rceil$ bits.

-Each atypical sequence is transmitted uncoded by $\lceil nlog\,|\chi|\rceil$ bits.

-Before transmitting each sequence a flag bit is sent to show whether the sequence is typical or not.

This means that the average (expected) compression rate can be bounded as:

$$R < \frac{1+(1-\varepsilon)\{n(H(X)+\varepsilon)+1\}+\varepsilon\{nlog\,|\chi|+1\}}{n}, \tag{2.11}$$

which can be simplified to

$$R < (1-\varepsilon)H(X)+\varepsilon log\,|\chi| + \frac{2+\varepsilon-\varepsilon^2}{n}. \tag{2.12}$$

It is easy to prove that $H(X) \leq log\,|\chi|$, with equality achieved for uniform sources, i.e. when all sequences are equi-probable [3]. Therefore, (2.12) can be rewritten as:

$$R < H(X)+\varepsilon(log\,|\chi| - H(X)) + \frac{2+\varepsilon-\varepsilon^2}{n}. \tag{2.13}$$

As n becomes large, ε can be chosen arbitrarily small, i.e. R approaches the entropy (since for a lossless code R is always lower bounded by $H(X)$).

2.3 Random binning

In the previous section we observed that achieving the entropy is theoretically possible. After discovering this fact, researchers started searching for practical codes to achieve the entropy. As an intermediate step, it was proved that random codes can achieve entropy. This proof is formulated in [3] as the random binning method.

Generally by an (n, nR) random code we mean a code that to each block of the message assigns a codeword of length nR bits where each bit is drawn i.i.d. with $Pr(1) = 0.5$. In other words, in random coding the codeword bits (we also call them the *parities*) are unbiased. Random binning follows this idea very closely as explained below:

Code construction:

1- choose 2^{nR} distinct indexes, for example 1, 2, ..., 2^{nR}.

2-To each message sequence assign an index randomly and equi-probably. In other words index i is assigned to each sequence with probability 2^{-nR}.

3-All message sequences with a common index are called a *bin*.

4-Save the codebook (i.e. bins and their elements) at both encoder and decoder

Encoding: Transmit the index of the bin containing the message sequence.

Decoding: Search for a *unique* typical sequence in the addressed bin. If not found, report a decoding error.

Figure 2.2 shows an example of a random binning scheme. The set of all message sequences consists of 5 typical sequences (shown by $+$) and 11 atypical sequences (shown by $*$). These numbers are considered only for demonstration purposes. The random binning scheme divides the message sequences to 4 bins indexed by 00, 01, 10, and 11, respectively. Now let us consider the three following examples and see how the decoder works in these cases:

1- The message sequence is one of the typical sequences in bin 00: The encoder transmits 00. The decoder searches for a unique typical sequence in bin 00 but since there are 3 typical sequences in the bin, a decoding error occurs.

2-The message sequence is the typical sequence in bin 10: The encoder transmits 10. The decoder searches in bin 10 for a unique typical sequence and finds it. The decoding is successful in this case.

3-The message sequence is the atypical sequence in bin 11: The encoder transmits 11. The decoder looks for a unique typical sequence in bin 11. Although the sequence in bin 11 is unique, it is not typical. Therefore, a decoding error is reported.

From the above example we observe that for short block length systems, the random binning method has errors and the compression is not lossless. However, the authors in [3] prove that for a given rate $R > H(X)$, as the block length grows, the probability of error goes to zero and the scheme asymptotically achieves lossless compression. It is important

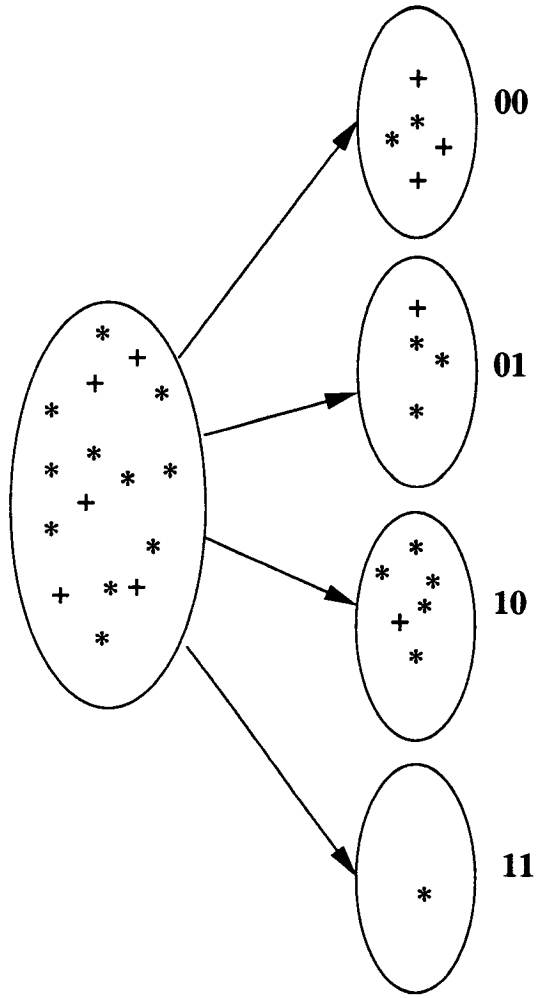

Figure 2.2: A random binning scheme with 16 message sequences and 4 bins. $*$ represents atypical sequences whereas $+$ represents typical sequences.

to notice that in random binning the probability of error is defined on the typical set. In fact according to the decoder's structure, the probability of error on the atypical set is one. However, by increasing the block length, probability of the atypical set, and therefore the probability of error on atypical set, becomes negligible (theorem 2.1, part 1).

The following theorem is taken from [3] and proves that for any given compression rate above the source entropy, random binning asymptotically achieves lossless compression:

Theorem 2.2- For any compression rate $R > H(X)$, by increasing n, the random

binning scheme becomes asymptotically lossless.

Proof- Let us show the message by a vector $\mathbf{x}$. The encoder transmits the index of the bin containing $\mathbf{x}$. If $\mathbf{x}$ is an atypical sequence a decoding error occurs. Also, if $\mathbf{x}$ is a typical sequence but another typical sequence $\mathbf{x}'$ is in the same bin of $\mathbf{x}$, a decoding error occurs. Thus:

$$Pr\{Error\} = Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} + Pr(\mathbf{x} \notin A_\varepsilon) Pr\{Error|\mathbf{x} \notin A_\varepsilon\}. \tag{2.14}$$

For sufficiently large n, we know $Pr(\mathbf{x} \notin A_\varepsilon) \leq \varepsilon$. Also from the decoder's structure we have $\lim_{n\to\infty} Pr\{Error|\mathbf{x} \notin A_\varepsilon\} = 1$. Therefore, the second term in the righthand side of (2.14) can be bounded as:

$$Pr(\mathbf{x} \notin A_\varepsilon) Pr\{Error|\mathbf{x} \notin A_\varepsilon\} \leq \varepsilon. \tag{2.15}$$

For the first term in righthand side of (2.14) we have:

$$Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} = \sum_{\mathbf{x}\in A_\varepsilon} \sum_{\mathbf{x}'\in A_\varepsilon, \mathbf{x}'\neq\mathbf{x}} Pr(\hat{\mathbf{y}} = \mathbf{y}) p(\mathbf{x}), \tag{2.16}$$

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the codewords assigned to $\mathbf{x}$ and $\mathbf{x}'$, respectively. Now (2.16) can be rewritten as:

$$Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} = \sum_{\mathbf{x}'\in A_\varepsilon} \left(Pr(\mathbf{y}' = \mathbf{y}) \sum_{\mathbf{x}\in A_\varepsilon} p(\mathbf{x} \in A_\varepsilon) \right). \tag{2.17}$$

Since in random binning indexes (codewords) are assigned randomly and equiprobably, we have $Pr(\mathbf{y}' = \mathbf{y}) = 2^{-nR}$. Also $\sum_{\mathbf{x}\in A_\varepsilon} p(\mathbf{x}) \leq 1$, which leads to:

$$Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} \leq \sum_{\mathbf{x}' \in A_\varepsilon} 2^{-nR}. \tag{2.18}$$

Finally we know that $\sum_{\mathbf{x}' \in A_\varepsilon} 2^{-nR} = 2^{-nR}|A_\varepsilon|$. Also from (2.10) we know that$|A_\varepsilon| \leq 2^{n(H(X)+\varepsilon)}$. Thus, the first term in righthand side of (2.14) can be bounded as follows:

$$Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} \leq 2^{-n(R-H(X)-\varepsilon)}. \tag{2.19}$$

For any compression rate $R > H(X) + \varepsilon$, by increasing n, the righthand side of (2.19) decreases. This means, given sufficiently large n, this term could become less than ε, and we will have:

$$Pr(\mathbf{x} \in A_\varepsilon) Pr\{Error|\mathbf{x} \in A_\varepsilon\} \leq \varepsilon, \tag{2.20}$$

provided that n is sufficiently large. Substituting (2.15) and (2.20) in (2.14) we conclude:

$$Pr\{Error\} \leq 2\varepsilon,$$

for sufficiently large n. Notice that ε may become small enough by increasing n. This means $R > H(X)+\varepsilon$ can be arbitrarily chosen close to $H(X)$ and $Pr\{Error\} \leq 2\varepsilon$ can become arbitrarily small, provided large enough n. In other words given a compression rate arbitrarily close to the entropy, the random binning scheme is asymptotically lossless. This completes the proof.

2.4 Lossless Turbo Source Coding

In the previous section we observed that large block length random codes could achieve the limit of lossless compression. As a result, random-like codes present themselves as good candidates for lossless compression. By random-like codes we mean codes with

constructive encoding and decoding structures that are capable of generating random-like codewords and performing near optimal decoding. Turbo codes, low density parity check (LDPC) codes, and repeat-accumulate (RA) codes are examples of these random-like codes. All these codes are originally designed for error correction (channel coding) but they could also be used for source coding. In this section we review the research on lossless data compression using turbo codes. Notice that there have been much works on compressing two correlated sources using turbo codes (the Slepian-Wolf model [11]) but we shall not consider them since our focus is on compressing a single source. For compression of correlated sources see [12]-[21] and references therein.

Compression of a single source using turbo codes is first considered in [27] where a near-lossless compression scheme is developed. Later a completely lossless scheme is presented in [22]. The structures of source encoders proposed in [27] and [22] are presented in this section. Then we explain the structure of source decoder which is the same for both schemes. Numerical results from the two references are given at the end.

2.4.1 Near-lossless and Completely Lossless Turbo Source Encoder

The near-lossless turbo source encoder is originally presented in [27]. The block diagram of this near-lossless encoder is shown in Fig. 2.3. First the message $\mathbf{x}$ is encoded using a turbo code (consisting of two parallel concatenated convolutional codes and a code interleaver as shown in Fig. 2.3). Then the parities are interleaved using a parity interleaver and are punctured to form a codeword $\mathbf{y}$. This scheme is a fixed-to-fixed length compression scheme, i.e. the input, $\mathbf{x}$, and the output, $\mathbf{y}$, both have fixed lengths. The main advantage of this scheme is its simple encoding, however the drawback is that a completely lossless compression cannot be guaranteed. More precisely, choosing a proper length for $\mathbf{y}$ (say l bits) ensures removing most of errors and giving a good estimate of the message, yet a small error probability is still inevitable. One solution to decrease this error probability is by increasing the number of decoding iterations [27].

Notice that there is no fundamental difference between code interleaver and parity interleaver. We just name them differently to emphasize that one is used as part of turbo code to interleave message bits whereas the other is used for interleaving the coded (parity) bits.

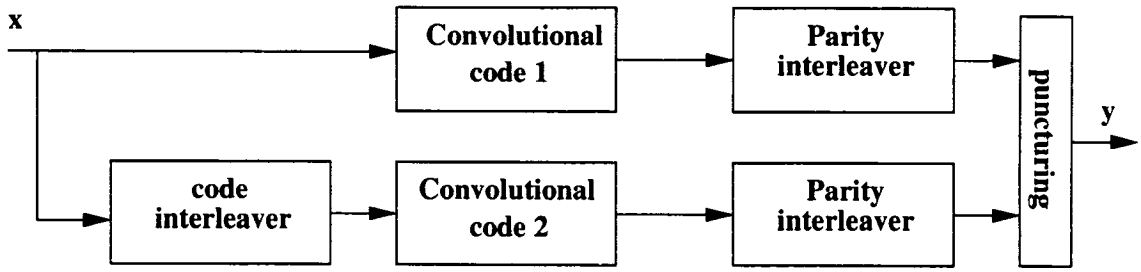

Figure 2.3: The block diagram of a near-lossless turbo source encoder

To guarantee a completely lossless compression, another turbo source coding scheme is proposed in [22]. The completely lossless scheme is similar to the near-lossless scheme with the difference that the puncturing block in Fig. 2.3 is replaced by an *iterative puncturing* block. The iterative puncturing block consists of a source decoder which performs tentative decoding and a comparator which compares the tentative reconstructed message with the original message. The iterative puncturing process is as follows:

Initialization: Puncture half of the parities.

Iteration:

1-Puncture $2\sqrt{n}$ more parities ($\sqrt{n}$ from each branch)[1].

2- Decode the codeword consisting of non-punctured parities.

3- Compare the decoding result with the original message.

4- If there is no error (i.e. decoding gives exactly the original message) return to 1.

5- Add the last $2\sqrt{n}$ punctured parities to the codeword. Transmit the codeword y and the codeword length l to the decoder.

Notice that in the presented completely lossless compression scheme the idea is

[1]Assume that we choose the block length n such that $\sqrt{n}$ is an integer.

to place a decoder along with the encoder and check if decoding will be successful (before transmission). This will take off the advantage of having a simple encoding process (because of checking and iterations) but will guarantee completely lossless compression. Since the scheme is now fixed-to-variable length (fixed-length input but variable-length output) the codeword length must be sent to the decoder. However, this codeword length is an integer between 0 and n which is divisible by $2\sqrt{n}$, thus it could be sent by $\left\lceil log_2 \frac{\sqrt{n}}{2} \right\rceil$ bits, i.e. the smallest integer number of bits which is greater than or equal to $log_2 \frac{\sqrt{n}}{2}$. For large n, transmitting these bits slightly increases the compression rate. For example for $n = 10000$, only 6 bits are required to represent the codeword length at the decoder. This means the compression rate increases by 0.0006 bps. Knowing the codeword, the decoder will be able to reconstruct the message. In fact, this has already been checked by the encoder prior to transmission.

2.4.2 Turbo Source Decoding and Numerical Results

We begin by defining the concept of L-value.

Definition: An L-value for a binary random variable x is defined by $log \frac{Pr(x=1)}{Pr(x=0)}$.

The turbo source decoding process is the same for both near-lossless and completely lossless turbo source coding schemes. Since both [27] and [22] are dealing with i.i.d. binary biased sources with $Pr(1) = p$, we explain the turbo source decoding for this type of sources. The block diagram of a turbo source decoder is shown in Fig. 2.4. The interleaver in Fig. 2.4 is the same as code interleaver in Fig. 2.3. The deinterleaver does the inverse function. In Fig. 2.4, $\mathbf{L}$ is the L-value representing the source *a priori* knowledge, i.e. $\mathbf{L} = log \frac{p}{1-p}$. $\mathbf{Z}_j$, $\mathbf{A}_j$, $\mathbf{E}_j$, and $\mathbf{D}_j$, $j = 1, 2$, are also L-values. $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are the L-values related to parities from the two convolutional codes. Each element of $\mathbf{Z}_1$ and $\mathbf{Z}_2$ is zero if its related parity bit is punctured, is $+\infty$ if its related bit is intact and is $+1$, and is $-\infty$ if its related bit is intact and is -1. $\mathbf{A}_1$ represents the *a priori* L-values for the first decoder. The first constituent decoder takes $\mathbf{L}$, $\mathbf{P}_1$ and $\mathbf{A}_1$ and calculates soft value $\mathbf{D}_1$ using the BCJR algorithm [22] (notice that $\mathbf{A}_1$ is not available in the first

iteration). Then extrinsic information on the systematic bits, $\mathbf{E}_1$, is expressed as:

$$\mathbf{E}_1 = \mathbf{D}_1 - \mathbf{A}_1 - \mathbf{L}, \tag{2.21}$$

$\mathbf{E}_1$ is interleaved and passed to the second decoder as the *a priori* input $\mathbf{A}_2$. The second decoder takes $\mathbf{L}$, $\mathbf{P}_2$ and $\mathbf{A}_2$, and feeds back extrinsic information $\mathbf{E}_2 = \mathbf{D}_2 - \mathbf{A}_2 - \mathbf{L}$ which becomes the *a priori* knowledge of the first decoder (after deinterleaving). This process iterates until convergence is achieved or a maximum number of iterations is reached (See [32] for details on turbo decoding algorithms).

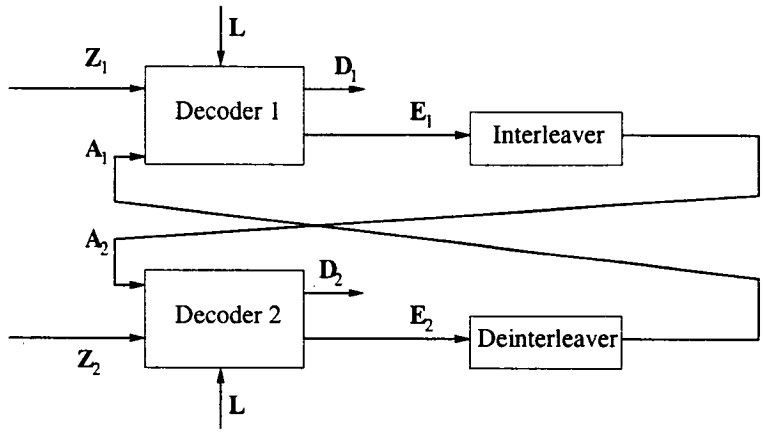

Figure 2.4: Block diagram of the turbo source decoder.

Table 2.1 shows the compression rates achieved by near-lossless (R_1) and completely lossless (R_2) schemes. For the near-lossless scheme the block length is 16384 bits and for the completely lossless case the block length is 10000 bits. It is observed that in some cases the completely lossless scheme achieves both better compression rate and zero error probability, however this comes at the expense of more encoding complexity. In other words the completely lossless scheme has performed several encoding and decoding iterations but the near-lossless scheme has just measured the error probability for different compression rates and has fixed the compression rate to the smallest one which achieves an error probability below a defined threshold. The threshold considered in [27] is a bit error rate equal 10^{-6}.

Table 2.1: The compression rates achieved by near-lossless (R_1) and completely lossless (R_2) turbo source coding schemes, compressing a binary i.i.d. source with $Pr(1) = p$.

p	$H(X)$	R_1	R_2
0.01	0.081	0.16	0.25
0.05	0.286	0.38	0.44
0.10	0.469	0.58	0.59
0.15	0.610	0.75	0.71
0.20	0.721	0.87	0.80

2.5 Conclusion

In this chapter we presented the fundamental concepts to be used in our work. We reviewed the basic definition of source coding and presented the main parameters of any source coding scheme. Then we reviewed the concept of typical sequences and introduced entropy as a limit for achievable rate of a lossless compression scheme. After that we showed that random codes are capable of achieving this limit, using a random binning scheme. To search for constructive methods to achieve competitive compression rates, we studied the performance of turbo codes for this purpose. It was observed that whether used for near-lossless or completely lossless compression, turbo codes are able to achieve small compression rates. Near-lossless turbo source coding offers a very simple and low-complexity encoding method, whereas completely lossless turbo source coding guarantees a zero error probability.

Chapter 3

Lossless Source Coding Using Nested Error Correcting Codes

In this chapter, we consider the theoretical performance of nested error correcting codes for data compression. For this, we generalize the conventional random binning argument to the so called *tree structured random binning* concept. We derive the distribution of the compression rate achieved by the proposed tree structured random binning scheme. Comparing this distribution with the distribution achieved using a library of random binning schemes, we prove that a nested code can achieve rates close to a library of codes but with much lower encoding/decoding complexity. Furthermore, for large data blocks, the proposed scheme is shown to asymptotically achieve the entropy limit.

3.1 Introduction

The application of error correcting codes to data compression was first introduced by Shannon [2] and was later considered in many references, including [23]-[31] (related work on joint compression of correlated sources could be found in [12]-[21] and references therein). In [2], Shannon observed that source coding and channel coding are information theoretic duals of each other. This duality can be used to employ a channel

code as a source code (or vice versa). In fact, one can use a random binning method [3] (will be discussed in Section 3.2) to show that random codes could be used for lossless data compression. In that sense, it was shown that by increasing the data block length, these random codes asymptotically achieve the entropy limit. Therefore, a random-like code (e.g. turbo code or low density parity check (LDPC) code) could be employed as a source code. For example, in [27], a turbo code is used to compress a biased binary independent identically distributed (i.i.d.) source. The encoder, in [27], generates two parity sequences using two parallel concatenated convolutional codes. Compression is achieved by heavily puncturing these parities.

The random binning argument presented in [3] considers a fixed-to-fixed length source coding scheme, i.e. a fixed-length message block is mapped to a fixed-length codeword. Although fixed-to-fixed length schemes asymptotically achieve the entropy, fixed-to-variable (or variable-to-fixed) length codes achieve lower compression rates in finite block length systems. To design a fixed-to-variable length code, one may use a library of parallel random binning schemes with different rates. As a practical implementation, [26] considers lossless compression of a biased *i.i.d.* binary source using a Hamming code, a BCH code and a rate-one code (see Section 3.3). As an alternative to using a library of codes, a fixed-to-variable length source code can be implemented by a nested code, where each codeword of a higher rate code is generated by adding bits to a codeword of a lower rate code. For example, fixed-to-variable length LDPC and turbo source coders have been designed in [29] and [22] respectively (alternative schemes based on fountain and repeat-accumulate (RA) codes are presented in [30] and [31] respectively). In [22], after encoding the message by a turbo code, the parities are gradually punctured using an iterative algorithm. Puncturing continues as long as the codeword is decodable with no errors. However, in [28], after transmitting the syndrome of the message, the message bits are sent one by one using an iterative doping approach until the transmitted bits are sufficient to decode the entire message at the decoder. If we refer to the method of [22] as a decremental redundancy method, the method in [28] can be

referred to as an incremental redundancy method.

In this chapter we consider the theoretical performance of nested error correcting codes for data compression. We generalize the conventional random binning argument to the so called tree structured random binning concept. Then we derive the distribution of the compression rate achieved by the proposed tree structured random binning scheme. Comparing this distribution with the distribution achieved using a library of random binning schemes, we prove that a nested code can achieve rates close to a library of codes but with much lower encoding/decoding complexity. Furthermore, for large data blocks, the proposed scheme is shown to asymptotically achieve the entropy limit.

3.2 Review of Random Binning

In source coding the input sequence and the output sequence of a source encoder could be either fixed length or variable length sequences. In other words if the input sequence consists of n samples and the output sequence consists of l bits, both n and l (or one of them) could be random variables. Therefore, four different types of source coding schemes are defined as follows:

(i) fixed-to-fixed length scheme: a scheme for which both n and l are constants.

(ii) fixed-to-variable length scheme: a scheme for which n is constant but l is a random variable.

(iii) variable-to-fixed length scheme: a scheme for which n is a random variable but l is constant.

(iv) variable-to-variable length scheme: a scheme for which both n and l are random variables.

In this chapter we focus on fixed-to-fixed and fixed-to-variable length schemes. We first review the concept of random binning as a fixed-to-fixed scheme. Then we present a fixed-to-variable length scheme, called tree structured random binning, as an extension of random binning.

The idea of lossless compression using the random binning procedure was introduced in [3] to prove that fixed-to-fixed length coding schemes may asymptotically achieve the entropy. To illustrate, let us consider a discrete i.i.d. source X that takes its outcome x from a finite alphabet χ with cardinality $|\chi|$. Also let n be the message block length in symbols, R be the compression rate in bits per sample, and call the space of all n tuples $\mathbf{x} = x_1 x_2 ... x_n$, $x_i \in \chi$, as χ^n with cardinality $|\chi|^n$. For a fixed $\varepsilon > 0$, we denote the ε-typical set [3] by A_ε, such that for each sequence $\mathbf{x} \in A_\varepsilon$,

$$2^{-n(H(X)+\varepsilon)} \leq Pr(\mathbf{x}) \leq 2^{-n(H(X)-\varepsilon)}. \tag{3.1}$$

It can be shown [3] that for any arbitrary value of ε, $Pr\{A_\varepsilon\} > 1 - \varepsilon$, provided that n is sufficiently large. In this case, the number of typical sequences satisfies the following inequality [3]:

$$(1 - \varepsilon)2^{n(H(X)-\varepsilon)} \leq |A_\varepsilon| \leq 2^{n(H(X)+\varepsilon)}. \tag{3.2}$$

Given this, the basic idea of random binning for lossless compression of X is to choose 2^{nR} indexes[1] (say 1 to 2^{nR}) and randomly and equi-probably assign an index to each of the $|\chi|^n$ possible outcomes. The set of all sequences with a common index is called a bin, and the encoder transmits the index of the bin containing the message sequence. In the decoding process, the decoder searches for a unique typical sequence in the addressed bin. If the message sequence is not typical, or if there is at least one other typical sequence in its related bin, a decoding error occurs.

Let us consider two typical sequences $\mathbf{x}$ and $\mathbf{x}'$. The probability that $\mathbf{x}$ does not fall in a common bin with $\mathbf{x}'$ is $(1 - 2^{-nR})$. Thus, the probability that $\mathbf{x}$ does not fall in a common bin with any other typical sequence is $(1 - 2^{-nR})^{|A_\varepsilon|-1}$. Therefore, the decoding error probability averaged over all possible random binning realizations is bounded by the following inequality,

[1] For simplicity we assume nR is a non-negative integer.

$$\overline{P_e} < \varepsilon + (1 - (1 - 2^{-nR})^{|A_\varepsilon|-1}). \tag{3.3}$$

In [3] it is shown that for sufficiently large n, and any $R > H(X) + \varepsilon$, $P_e \leq 2\varepsilon$. A similar result can be obtained here by noticing that $(1 - 2^{-nR})^{|A_\varepsilon|-1} \geq 1 - (|A_\varepsilon| - 1)2^{-nR}$. Therefore, from (3.3):

$$\overline{P_e} < \varepsilon + (|A_\varepsilon| - 1)2^{-nR}. \tag{3.4}$$

Using (3.2), (3.4) can be rewritten as:

$$\overline{P_e} < \varepsilon + 2^{n(H(X)+\varepsilon)}2^{-nR}. \tag{3.5}$$

For $R > H(X) + \varepsilon$, $2^{n(H(X)+\varepsilon)}2^{-nR}$ is a monotonically decreasing function of n and its value becomes smaller than ε for sufficiently large n. Therefore for large values of n, from (3.5), we can see that $\overline{P_e} < 2\varepsilon$. That is for large enough block lengths, the rate can be arbitrarily close to the source entropy, $H(X)$, while the decoding error probability averaged over the codes' ensemble is negligible. Thus, at least one code of this ensemble is able to compress the source with negligible error, while its rate approaches the source entropy as the block length increases. For practical implementations of random binning, one may use random-like codes e.g. LDPC or turbo codes, where the codeword is generated using a syndrome-former [29] or alternatively as the parities of a non-systematic punctured code [27]. In this case, the typical set decoding can be implemented by near maximum likelihood decoding of these codes.

3.3 Performance Evaluation of a Library of Random Binning Schemes

To achieve lower compression rates using finite-length codes, and ensure completely lossless compression (not only over the typical set but over all message sequences), fixed-to-fixed length source coding schemes can be replaced by fixed-to-variable length schemes. In this case, the message length is fixed but the codeword length can change from one codeword to another. As an example of fixed-to-variable length codes, a library of M random binning schemes can be used in parallel where $R_1 < R_2 < ... < R_M$ with $R_i \leq \lceil \log |\chi| \rceil$ being the compression rate of the ith scheme (for simplicity we assume that nR_i is an integer for all values of i). The encoder in this case works as follows. The message[2] is encoded using code 1 and its decodability is tested (at the encoder). If it is decodable with no error (i.e. if it is the only typical sequence in the bin), the related codeword is transmitted. Otherwise the message is encoded using code 2, 3, ..., M until the first successful decoding is reached. Note that as long as $R_M > H(X)$, lossless compression is guaranteed for sufficiently large n. Also, completely lossless compression can be provided by adding an uncoded scheme to the library.

Now, let us derive the distribution of the compression rate for this parallel random binning scheme. To proceed, suppose we are given a typical sequence x and would like to find the probability that x is encoded to rate R_j. Recall that given a rate R_i, the probability that x does not fall in a common bin with any other typical sequence is $(1 - 2^{-nR_i})^{|A_\epsilon|-1}$. Thus, the probability that x cannot be encoded to a rate $R_i < R_j$ is $1 - (1 - 2^{-nR_i})^{|A_\epsilon|-1}$. Also the probability that x can be encoded to rate R_j is $(1 - 2^{-nR_j})^{|A_\epsilon|-1}$. Therefore, the probability that x is encoded to rate R_j is:

$$Pr_{\mathbf{x}}(R_j) = (1 - 2^{-nR_j})^{|A_\epsilon|-1} \prod_{i=1}^{j-1} \{(1 - (1 - 2^{-nR_i})^{|A_\epsilon|-1}\}. \tag{3.6}$$

[2]We assume that the message is a typical sequence. To encode atypical sequences, an uncoded scheme with rate $\lceil log_2 |\chi| \rceil$ bps can be added to the library.

If we let n go to infinity, then $(1-2^{-nR_i})$ is equivalent to $exp(-2^{-nR_i})$ and $(1-2^{-nR_i})^{|A_\varepsilon|-1}$ is equivalent to $exp(-(|A_\varepsilon|-1)2^{-nR_i})$; which can be rewritten as $exp(-2^{-n(R_i-\frac{log_2(|A_\varepsilon|-1)}{n})})$. Thus, if $R_i > \frac{log_2(|A_\varepsilon|-1)}{n}+\varepsilon$, $(1-2^{-nR_i})^{|A_\varepsilon|-1}$ tends to one by increasing n. Also if $R_i < \frac{log_2(|A_\varepsilon|-1)}{n}-\varepsilon$, then $(1-2^{-nR_i})^{|A_\varepsilon|-1}$ tends to zero $(1-(1-2^{-nR_i})^{|A_\varepsilon|-1}$ tends to one) by increasing n. Furthermore, for sufficiently large values of n, $H(X)-\varepsilon < \frac{log_2|A_\varepsilon|}{n} < H(X)+\varepsilon$ (obtained by taking log_2 from all sides of (3.2) and neglecting $\frac{log_2(1-\varepsilon)}{n}$ as n goes to infinity). For any $|A_\varepsilon| > 2$, it is easy to show that $log_2|A_\varepsilon|-1 < log_2(|A_\varepsilon|-1) < log_2|A_\varepsilon|$. By dividing all sides by n and recalling $H(X)-\varepsilon < \frac{log_2|A_\varepsilon|}{n} < H(X)+\varepsilon$, we conclude $H(X)-\varepsilon < \frac{log_2(|A_\varepsilon|-1)}{n} < H(X)+\varepsilon$ (neglecting $\frac{-1}{n}$ as n goes to infinity). Therefore, $R_i < H(X)-2\varepsilon$ is a sufficient condition for $R_i < \frac{log_2(|A_\varepsilon|-1)}{n}-\varepsilon$ and $R_i > H(X)+2\varepsilon$ is a sufficient condition for $R_i > \frac{log_2(|A_\varepsilon|-1)}{n}+\varepsilon$. From the above discussion and using (3.6):

$$\lim_{n\to\infty} Pr(R_i) = 0;\ \forall R_i:\ R_i < H(X)-2\varepsilon,\ or\ R_{i-1} > H(X)+2\varepsilon. \tag{3.7}$$

One can justify (3.7) as follows. From (3.6), notice that if $R_i < H(X)-2\varepsilon$, $(1-2^{-nR_i})^{|A_\varepsilon|-1}$ goes to zero and hence $Pr(R_i)$ goes to zero. Similarly if $R_{i-1} > H(X)+2\varepsilon$, $(1-(1-2^{-nR_{i-1}})^{|A_\varepsilon|-1})$ goes to zero and hence $Pr(R_i)$ goes to zero. Now assume that the set of rates $R_1 < R_2 < ... < R_M$ is given such that for a unique index j, $R_j > H(X)+2\varepsilon$, and $R_{j-1} < H(X)-2\varepsilon$. Then using (3.7), as n goes to infinity, $Pr(R_j)$ goes to one and $Pr(R_{i\neq j})$ goes to zero. This implies that by increasing the block length, the rate distribution tends to a discrete delta function in R_j where R_j is the smallest available rate greater than $H(X)+2\varepsilon$. That is, for sufficiently large block lengths, R_j can be chosen arbitrarily close to the entropy. Notice that (3.6) represents the rate distribution only for typical sequences. For atypical sequences, the rate distribution is considered as a discrete delta function at $R = \lceil log_2|\chi|\rceil$ (uncoded scheme). This event occurs with probability less than or equal to ε which could become arbitrarily small by increasing n [3]. Also notice that when x is encoded using a library of codes, transmitting the code index is not

necessary. Moving on to the decoding process, nR_1 bits are decoded using code 1. If decoding fails (the indexed bin in code 1 does not contain a unique typical sequence), the decoder adds $n(R_2 - R_1)$ bits and then examines the decodability using code 2. This process continues until the first successful decoding is reached. Note that in any case, succesful decoding can be guaranteed by adding an uncoded scheme to the library.

The practical implementation of this parallel random binning scheme was examined by Ancheta [26] where a rate zero code (an all zero codeword), a Hamming code, a BCH code, and a rate one code (uncoded transmission) were used in parallel to compress messages of block length 15 for a biased binary *i.i.d.* source. Similar to [26], [29] and [30] have proposed different compression schemes using different library of codes to achieve compression rates close to the source entropy[3]. It is important to mention that the improvement achieved using these codes, relative to the single-code case, is at the expense of large encoding/decoding complexity.

3.4 Lossless Source Coding using Tree Structured Random Binning

As an alternative to using a library of codes, one can use a nested code where each codeword of higher rate code is formed by adding some bits to a codeword of lower rate code. Later, we show that this is equivalent to a tree structured random binning scheme where each bin is split into smaller bins until each bin contains no more than one typical sequence. For practical implementations of this tree structured random binning scheme, the reader is referred to [22]-[31]. In [22], ([31]), after encoding the information bits by a turbo code (RA code), the parities are gradually punctured using an iterative algorithm. The puncturing process continues as long as the codeword is decodable with no error. However, in [29] and [30] after transmitting the syndrome of the message, the message

[3]In [29] and [30] the ideas of using a nested code and a library of codes are combined to further reduce the compression rate. In other words, a library of nested codes is used.

bits are sent one by one using an iterative doping approach until the transmitted bits are sufficient to decode the entire message at the decoder.

Now we introduce our tree structured random binning scheme and show that it asymptotically achieves the entropy.

Code construction: To construct the code, an index $y \in \{1, 2, .., 2^m\}$, for some integer $m \geq 1$, is assigned to each message sequence, randomly and equi-probably. All message sequences with a common index form a bin. Each bin containing more than one typical sequence is split into 2^m smaller bins. The splitting process for each bin continues until no bin contains more than one typical sequence.

Encoding: The encoder observes an n-bit message block $\mathbf{x}$, and transmits the address of that bin containing $\mathbf{x}$.

Decoding: The decoder searches for a unique typical sequence in the addressed bin. If $\mathbf{x}$ is not typical, a decoding error occurs.

Since the splitting process for some bins may terminate sooner than others, the constructed code has a variable length output. Fig. 3.1 shows an example of a tree structured random binning scheme for 16 sequences. The number of typical sequences is assumed to be 5 (only for demonstration purposes).

Let us refer to the ε typical set [3] of the message sequences by A_ε (with cardinality $|A_\varepsilon|$). It is known [3] that for sufficiently large n:

$$Pr(\mathbf{x} \in A_\varepsilon) > 1 - \varepsilon, \tag{3.8}$$

and also:

$$(1 - \varepsilon)2^{n(H(X)-\varepsilon)} \leq |A_\varepsilon| \leq 2^{n(H(X)+\varepsilon)}. \tag{3.9}$$

For $i \geq 0$, let N_i be a non-negative integer and define $N_0 = |A_\varepsilon| - 1$. Assume that the bin containing $\mathbf{x}$ is split $l - 1$ times ($l \geq 1$) and $\mathbf{x}$ is in a common bin with N_{l-1} other typical sequences, where $N_{l-1} > 0$. Since $N_{l-1} > 0$, the bin is split into 2^m smaller bins.

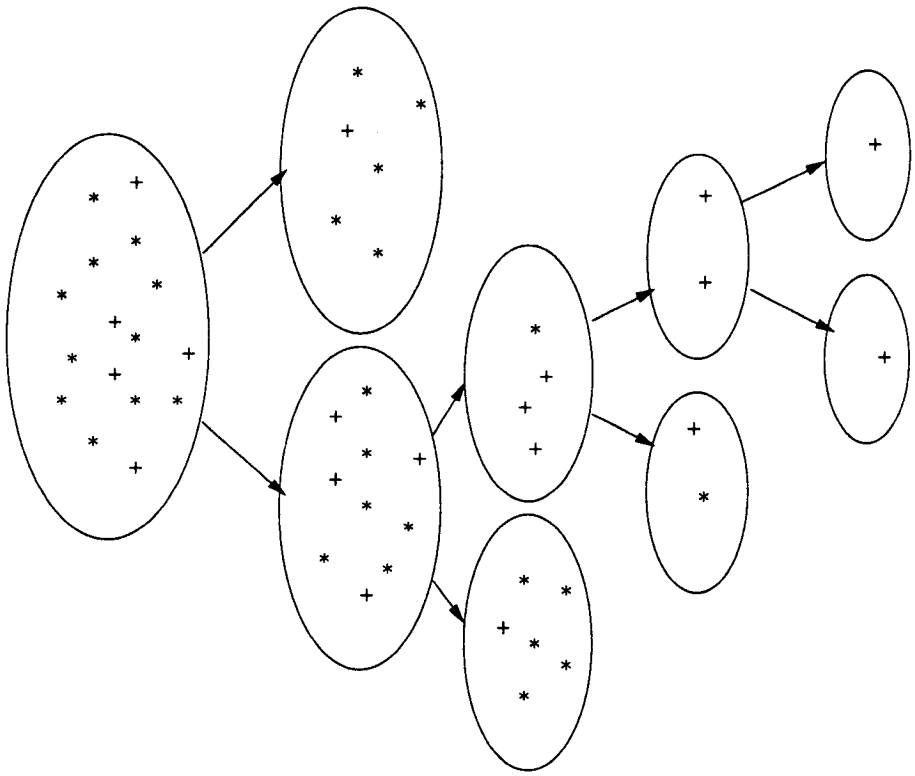

Figure 3.1: A tree structured random binning scheme with $m = 1$ for a source with 16 message sequences, including 5 typical sequences. Typical sequences are shown by + and atypical sequences are shown by *.

Then the probability that x falls in a common bin with N_l other typical sequences, where $0 \leq N_l \leq N_{l-1}$, is given by

$$Q_m(N_{l-1}, N_l) = \frac{\begin{pmatrix} N_{l-1} \\ N_l \end{pmatrix} (2^m - 1)^{N_{l-1}-N_l}}{2^{mN_{l-1}}}, \tag{3.10}$$

for any typical sequence x. To justify (3.10), note that x can fall in each of the 2^m bins and the N_l sequences that are in the same bin with x can be selected in $\begin{pmatrix} N_{l-1} \\ N_l \end{pmatrix}$ ways, with each of the $N_{l-1} - N_l$ other typical sequences can fall in each of the $2^m - 1$ other bins. If $N_l = 0$, the splitting process for the bin containing x is terminated and l is the length of the codeword assigned to x.

Now, if the length of the codeword assigned to x is denoted by l_x, this codeword can be expressed by ml_x bits where l_x is a random variable with the following probability mass function:

$$q_m(l_x = l) = \sum_{N_1=1}^{N_0} \sum_{N_2=1}^{N_1} \cdots \sum_{N_{l-1}=1}^{N_{l-2}} Q_m(N_{l-1}, 0) \prod_{i=1}^{l-1} Q_m(N_{i-1}, N_i). \tag{3.11}$$

For simplicity, we denote $q_m(l_x = l)$ by $q_m(l)$ in the sequel. Note that $q_m(l)$ is the probability that $N_i > 0$ for $i < l$ and $N_l = 0$. From (3.11), one can observe that $q_m(l)$ is a function of m, l, and N_0. For brevity, the dependency of $q_m(l)$ on N_0 is dropped in the notation.

Theorem 3.1- For any value of $N_0 > 0$ and for $l > 0$:

$$q_m(l) = (1 - 2^{-ml})^{N_0} - (1 - 2^{-ml+m})^{N_0}. \tag{3.12}$$

Proof :The proof of this theorem is by induction on $l \geq 1$. Consider a typical sequence x. The probability that the length of the codeword assigned to x is 1, $q_m(1)$, is given by

$$q_m(1) = Q_m(N_0, 0) = (1 - 2^{-m})^{N_0}. \tag{3.13}$$

Assume that for some l, (3.12) is valid. Then, from (3.11),

$$\mathrm{q}_m(l+1) = \sum_{N_1=1}^{N_0} \sum_{N_2=1}^{N_1} \cdots \sum_{N_l=1}^{N_{l-1}} Q_m(N_l, 0) \prod_{i=1}^{l} Q_m(N_{i-1}, N_i), \tag{3.14}$$

which can be rewritten as:

$$\mathrm{q}_m(l+1) = \sum_{N_1=1}^{N_0} Q_m(N_0, N_1) \times \sum_{N_2=1}^{N_1} \cdots \sum_{N_l=1}^{N_{l-1}} Q_m(N_l, 0) \prod_{i=2}^{l} Q_m(N_{i-1}, N_i). \tag{3.15}$$

From (3.11) and (3.12), it is easy to show that

$$\sum_{N_2=1}^{N_1} \cdots \sum_{N_l=1}^{N_{l-1}} Q_m(N_l, 0) \prod_{i=2}^{l} Q_m(N_{i-1}, N_i) = (1\text{-}2^{-ml})^{N_1} - (1 - 2^{-ml+m})^{N_1}. \quad (3.16)$$

Now, from (3.15) and (3.16),

$$q_m(l+1) = (1 - 2^{-m})^{N_0} \sum_{N_1=1}^{N_0} \begin{pmatrix} N_0 \\ N_1 \end{pmatrix} ((\tfrac{1-2^{-ml}}{2^m-1})^{N_1} - (\tfrac{1-2^{-ml+m}}{2^m-1})^{N_1}). \quad (3.17)$$

One can use the binomial expansion to show that

$$\sum_{N_1=1}^{N_0} \begin{pmatrix} N_0 \\ N_1 \end{pmatrix} (\frac{1 - 2^{-ml}}{2^m - 1})^{N_1} = (\frac{2^m - 2^{-ml}}{2^m - 1})^{N_0} - 1. \quad (3.18)$$

Hence using (3.18) and after simplification, (3.17) can be expressed as

$$q_m(l+1) = (1 - 2^{-ml-m})^{N_0} - (1 - 2^{-ml})^{N_0}. \quad (3.19)$$

In Fig. 3.2, we show the probability mass function (pmf), $q_m(l)$, for the codeword length of the tree structured random binning scheme with $N_0 = 2^{40}$ and $m = 1$. As shown, the expected value of l is 41.33 bits. Note that if one uses a library of random binning schemes with $l = \{1, 2, ...\}$, (i.e. $R_i = i/n$ for $i = 1, 2, ...$), from (3.12), the expected value of the codeword length can be found equal to 40.63 bits . The probability mass function using a library of random codes is illustrated in Fig. 3.3. As another comparison Fig. 3.4 and Fig. 3.5 show the pmf of compression rate with $N_0 = 2^{20}$, and $m = 1$, using tree structured random binning and a library of random binning schemes, respectively. It is observed that in both comparisons, the average compression length achieved by tree structured random binning scheme is 1.33 bits more than the minimum achievable length, i.e. $log_2 N_0$. Later we will show that this always holds for large values of N_0.

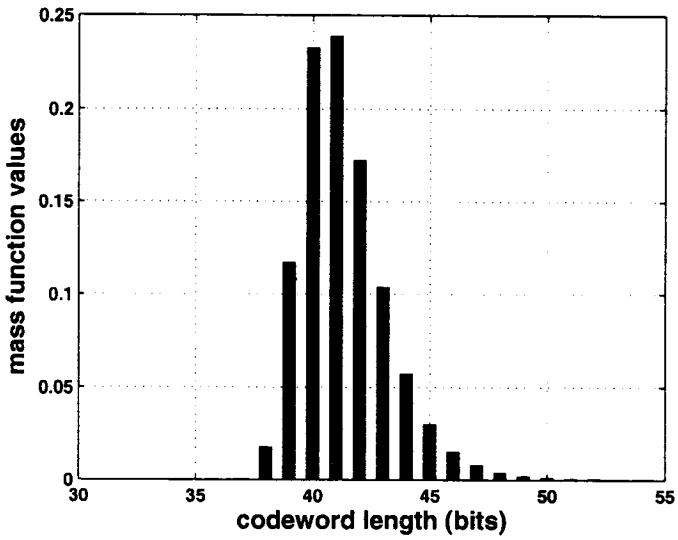

Figure 3.2: Probability mass function of the codeword length for a tree structured random binning scheme with $N_0 = 2^{40}$, and $m = 1$. $\bar{l} = 41.33$ bits.

From (3.12), recall that

$$q_1(l) = (1 - 2^{-l})^{N_0} - (1 - 2^{-l+1})^{N_0}. \tag{3.20}$$

The following theorem states two important properties of $q_1(l)$ in the extreme case when N_0 goes to infinity. Later, these properties will be used to show that the tree structured random binning scheme is entropy achieving.

Theorem 3.2. If N_0 goes to infinity, then:

1) For any finite value of l, $\lim_{N_0 \to \infty} q_1(l) = 0$, i.e. the probability of any codeword with finite length l is zero.

2) $q_1(l)$ is concentrated about $log_2(N_0)$, i.e. for any value of l such that $\frac{l}{log_2 N_0}$ is much smaller or larger than one, $q_1(l)$ tends to zero. In fact, $q_1(l)$ will not tend to zero if and only if $\frac{l}{log_2 N_0}$ goes to one.

Proof 1) For any finite value of l, from (3.20), $\lim_{N_0 \to \infty} q_1(l) = 0$.

2) Let $N_0 \to \infty$ and $l \to \infty$. Given this, $(1 - 2^{-l})$ is equivalent to $exp(-2^{-l})$ and

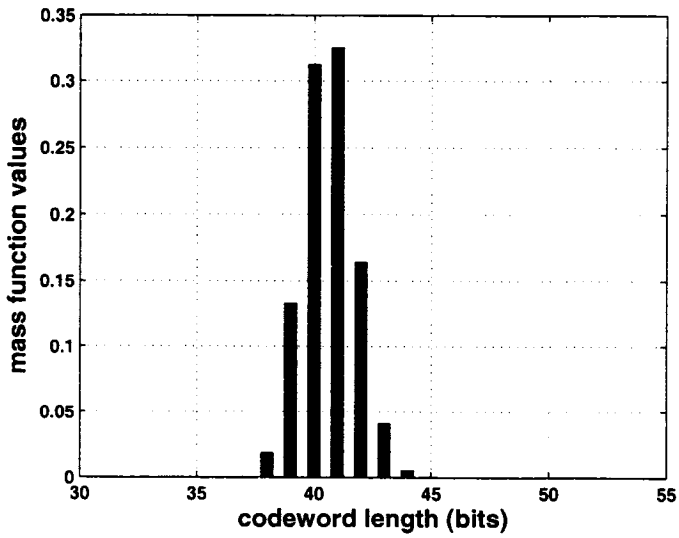

Figure 3.3: Probability mass function of the codeword length for a library of random binning schemes with $N_0 = 2^{40}$. $\bar{l} = 40.63$ bits.

from (3.20):

$$q_1(l) \rightarrow \exp(-N_0 2^{-l}) - \exp(-2N_0 2^{-l}). \tag{3.21}$$

Notice that the righthand side of (3.21) is non-zero only if for a finite value, c, $N_0 2^{-l} \rightarrow c$. Therefore, $log_2 N_0 - l \rightarrow log_2 c$ and finally $\frac{l}{log_2 N_0}$ converges to one.

Remark : Let us define a random variable $\beta = l - log_2 N_0$. Since $N_0 2^{-l} = 2^{-\beta}$, from (3.21), by increasing N_0, $q_1(l)$ tends to $f(\beta)$, where

$$f(\beta) = \exp(-2^{-\beta})(1 - \exp(-2^{-\beta})). \tag{3.22}$$

In other words, for large values of N_0, one can think of $q_1(l)$ as the function $f(\beta)$, shifted by $log_2(N_0)$ and sampled at positive integer points l. For $f(\beta)$, one can prove that $\int_{-\infty}^{+\infty} f(\beta)d\beta = 1$. Therefore, the non-negative function $f(\beta)$ represents a probability density function. Furthermore, calculation shows that $\int_{-\infty}^{+\infty} \beta f(\beta)d\beta = 1.333$. Given

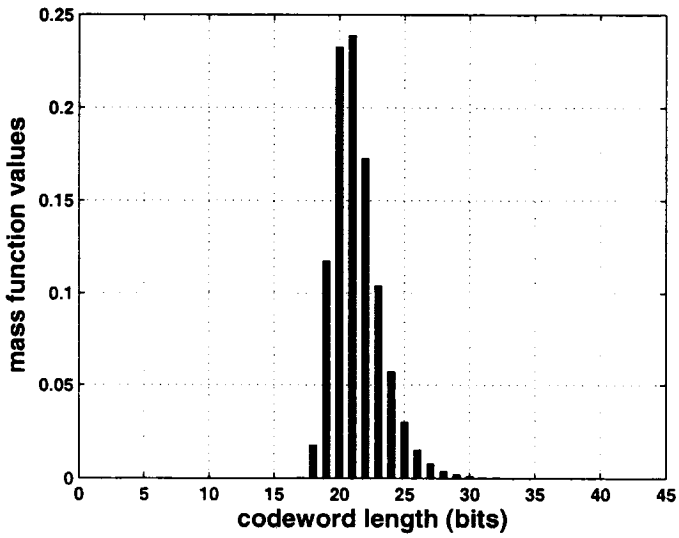

Figure 3.4: Probability mass function of the codeword length for a tree structured random binning schemes with $N_0 = 2^{20}$. $\bar{l} = 21.33$ bits.

the expected value of β, $\overline{\beta}$, the expected value of l can be represented as

$$\bar{l}_1 = log_2 N_0 + \overline{\beta}. \tag{3.23}$$

Now if we assume β to be continuous, one would have $\bar{l}_1 = log_2 N_0 + 1.333$. However, β is in fact discrete (because l is discrete) and its expected value, $\overline{\beta}$, is a function of N_0. Table 3.1 shows the value of $\bar{l}_1$ for different values of N_0.

Table 3.1: Expected value of codeword length, $\bar{l}_1$, versus $log_2 N_0$.

$log_2 N_0$	2	5	7	10	20	40
$\bar{l}_1$	3.505	6.355	8.338	11.333	21.333	41.333

Fig. 3.6 compares the average codeword length generated by tree structured random binning and a library of random binning schemes. It is observed that by increasing N_0, the average codeword length of a tree structured code converges to $log_2 N_0 + 1.333$ bits whereas the average codeword length of a library of codes converges to $log_2 N_0 + 0.630$ bits. Thus, a tree structured code is only 0.703 bits worse than a library of codes.

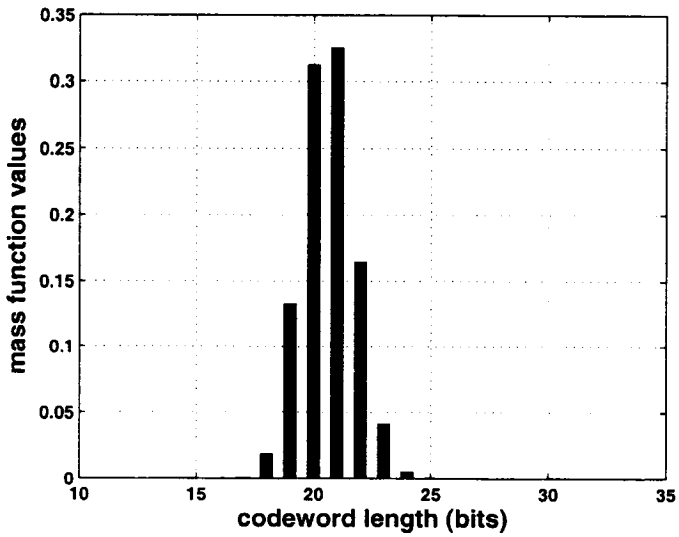

Figure 3.5: Probability mass function of the codeword length for a library of random binning schemes with $N_0 = 2^{20}$. $\bar{l} = 20.63$ bits.

In what follows, we first show that the proposed tree structured random binning scheme is entropy achieving for large block lengths and $m = 1$. Then we generalize our results to the case of $m > 1$. To proceed, let $\bar{l}_m$ be the expected value of l for a fixed m, i.e.

$$\bar{l}_m = \sum_{l=1}^{\infty} l.q_m(l). \tag{3.24}$$

Also, let $\overline{R}_b$ be the average compression rate achieved by the proposed random binning scheme, i.e.

$$\overline{R}_b = \frac{m}{n}\bar{l}_m. \tag{3.25}$$

One can see from (3.25) that $\overline{R}_b$ is a function of both m and n. From theorem 3.2 (part 2), and (3.24),

$$\lim_{N_0 \to \infty} \frac{\bar{l}_1}{log_2 N_0} = 1. \tag{3.26}$$

Since $N_0 = |A_\varepsilon| - 1$, from (3.9), $log_2 N_0 < n(H(X) + \varepsilon)$. Then, from (3.25) and (3.26):

$$\lim_{N_0 \to \infty} \overline{R}_b < H(X) + \varepsilon, \tag{3.27}$$

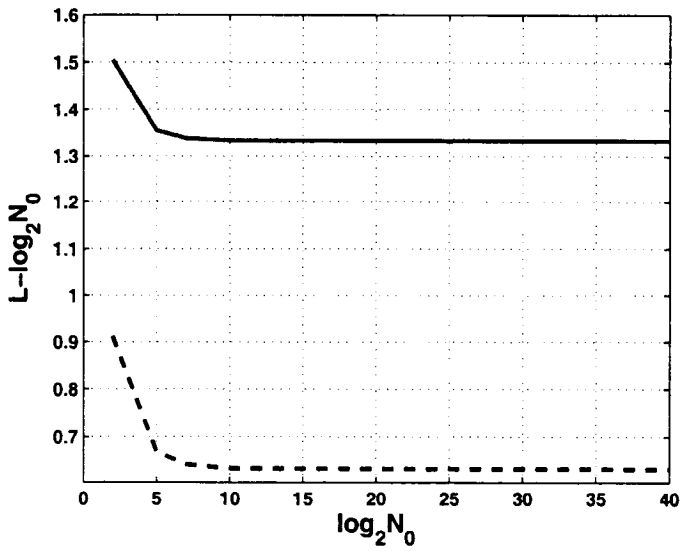

Figure 3.6: The difference between average codeword length and $log_2 N_0$ for a library of codes (Dashed) and a tree structured code (Solid).

for $m = 1$. Since ε could be arbitrarily small for large enough block lengths [3], one can conclude that the scheme is entropy achieving for $m = 1$ and large block lengths.

Now assume that an arbitrary typical sequence, $\mathbf{x}$, is encoded by a tree structured random binning code with $m > 1$ (Scheme 1). The length of the codeword assigned to $\mathbf{x}$ is l_m samples, where each sample consists of m bits. Therefore, this codeword can be described by ml_m bits. To reduce the compression rate, one may reduce m to $m = 1$ (Scheme 2); thus the codeword length may be decreased to l_1 bits, where $m(l_m - 1) < l_1 \leq ml_m$. More precisely, since Scheme 1 splits each bin to 2^m bins (adds m bits to the codeword at each step), Scheme 2 may do better by adding one bit to the codeword at each step and reducing the codeword length to l_1 bits. However, l_1 cannot be less than $m(l_m - 1)$, otherwise Scheme 1 would assign a codeword of length $m(l_m - 1)$ bits to $\mathbf{x}$. One may rewrite $m(l_m - 1) < l_1$ as $ml_m \leq l_1 + m - 1$, which leads to:

$$m\overline{l_m} \leq \overline{l_1} + m - 1. \tag{3.28}$$

From (3.25), (3.27), and (3.28) one can conclude that the scheme is entropy achieving for

any value of m and large enough n. Table 3.2 shows the value of $\bar{l}_m$for $N_0 = 2^{40}$ and different values of m.

Table 3.2: The value of $\bar{l}_m$for $N_0 = 2^{40}$ and different values of m.

m	$\bar{l}_m$
1	41.33
2	41.83
3	42.36
4	42.79
5	43.32
7	43.56

3.4.1 Completely lossless compression

To guarantee completely lossless compression, one has to encode all atypical sequences to rate one[4]. Note that, in this case, a flag bit has to be sent to identify whether the coded sequence is typical or atypical. If the sequence is typical, the address of the bin containing that sequence is transmitted after the flag bit. On the other hand if the sequence is atypical, the sequence itself is transmitted after the flag bit. If we denote the average compression rate for this system by $\widetilde{R_b}$, then

$$\widetilde{R_b} = (1 - \varepsilon_0)\overline{R_b} + \varepsilon_0 + \frac{1}{n}, \tag{3.29}$$

where ε_0 represents the value of ε for which the curves $Pr\{A_\varepsilon\}$ and $1 - \varepsilon$ intersect, i.e. $Pr\{A_{\varepsilon_0}\} = 1 - \varepsilon_0$[5]. Obviously for a large enough block length, ε_0 goes to zero and therefore $\lim_{N_0 \to \infty} \widetilde{R_b} = \lim_{N_0 \to \infty} \overline{R_b}$. For the case of finite block length systems, one has to analyze the behavior of ε_0 versus n to compute $\widetilde{R_b}$. To do this, we start by defining

$$g(\varepsilon) = Pr(\mathbf{x} \in A_\varepsilon) = Pr\left\{\left|-\frac{1}{n} logp(\mathbf{x}) - H(X)\right| < \varepsilon\right\}. \tag{3.30}$$

[4]In this subsection we restrict our study to completely lossless compression of binary sources. If the source alphabet contains 2^q elements, atypical sequences are sent at rate q bits per sample.

[5]ε_0 is uniquely found since $Pr\{A_\varepsilon\}$ is a non-decreasing function of ε and $1 - \varepsilon$ is a decreasing function of ε.

For a memoryless binary source, with $p = Pr\{x = 1\}$, it is easy to show that

$$g(\varepsilon) = \sum_{w=w_0}^{w_1} \binom{n}{w} p^w (1-p)^{n-w} \tag{3.31}$$

where

$$w_0 = \left\lceil n\frac{-\varepsilon + log(1-p) + H(X)}{log(1-p) - log(p)} \right\rceil - 1, \tag{3.32}$$

$$w_1 = \left\lfloor n\frac{\varepsilon + log(1-p) + H(X)}{log(1-p) - log(p)} \right\rfloor + 1, \tag{3.33}$$

where $\lfloor w_1 \rfloor$ represents the largest integer less than or equal to w_1. Note that for any value of n, $|A_\varepsilon| = \sum_{w=w_0}^{w_1} \binom{n}{w}$. As an example, Fig. 3.7 shows $g(\varepsilon)$ versus ε for $n = 4000$ bits where $g(\varepsilon)$ intersects with the line $1 - \varepsilon$ at $\varepsilon_0 = 0.0317$. In Fig. 3.8, we show the value of ε_0 as a function of the block length n.

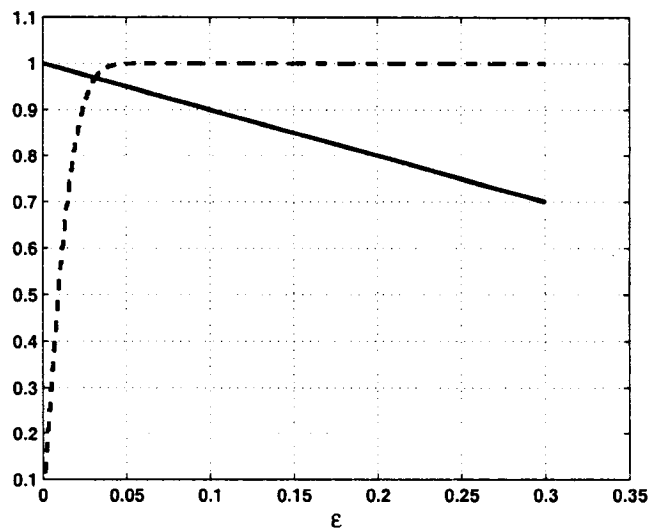

Figure 3.7: $g(\varepsilon)$ (dashed) versus ε for $n = 4000$. The solid line shows $1 - \varepsilon$.

Note that the minimum rate required for completely lossless compression can be obtained by finding a set A_ε to minimize $\overline{R_b} \times Pr(A_\varepsilon) + (1 - Pr(A_\varepsilon))$, where $\overline{R_b}$ is a

function of $|A_\varepsilon|$. One should also note that the value of $\widetilde{R}_b$ in (3.29) can be considered as an upper bound for the achievable rate of completely lossless compression.

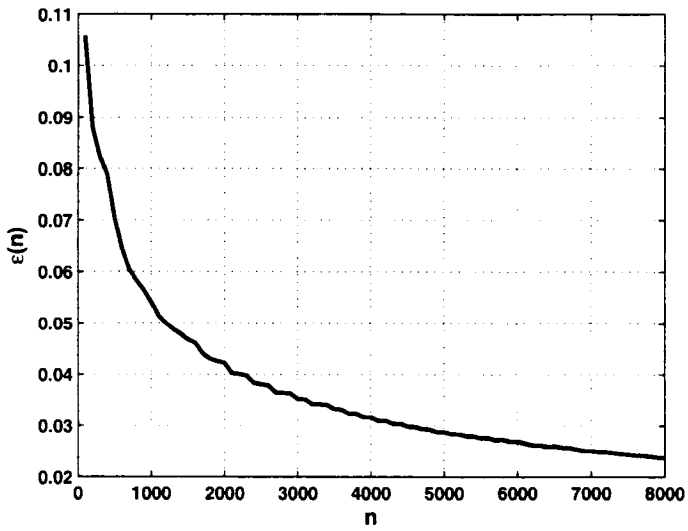

Figure 3.8: The value of ε_0 as a function of n.

3.4.2 Tree structured codes with biased parities

Dividing each bin to two bins in the tree structured random binning can be interpreted as adding unbiased independent random parities to the codeword until it can be distinguished from other codewords (in the typical set). This is possible when a random code is used. But when a practical code (like turbo or LDPC code) is used to generate the parities, these parities are always correlated (because of the code structure). For example when a convolutional code is used, the parities are correlated because of the trellis. It is possible to choose these parities far away from each other on the trellis so that the correlation decreases, but this correlation still exists. Sometimes there exists a one to one map from this parity sequence to another sequence which is uncorrelated (or less correlated), but that sequence would be biased. For example, assume we have a binary biased i.i.d. source X which generates independent bits x_i with $Pr\{x_i = 1\} = p$ for some $p < 0.5$. We use a single memory convolutional code with generator polynomial $G(D) = \frac{1}{1+D}$

to generate parity sequence $y_1y_2y_3...$, then puncture these parities and for some $k > 1$ transmit $y_ky_{2k}y_{3k}....$ The transmitted sequence can be mapped one to one, to the following sequence y_k, $y_{2k} + y_k$, $y_{3k} + y_{2k}, ...$ Also from the generator polynomial we have $y_i = y_{i-1} + x_i$. Thus for any integer $i \geq 0$, we will have

$$y_{ik} + y_{ik+k} = \sum_{j=ik+1}^{ik+k} x_j,$$

defining $y_0 = 0$. All additions are binary additions, i.e. they are exclusive or's. Since $x_i's$ are independent and biased, the mapped sequence is also an independent and biased sequence. In fact one could show that [3]:$Pr(y_{ik} + y_{ik+k} = 1) = \frac{1}{2}(1 - (1 - 2p)^k)$.

Regarding the above observation, we extend the results on the rate distribution of tree structured random binning as follows:

Assume that we are given $N_0 + 1$ message sequences. A tree structured scheme assigns codewords to these sequences in an iterative manner as follows. At the lth iteration, the lth bit of each codeword is generated independently with $Pr(1) = p$. After generating the lth bit for all codewords, the codewords assigned to all typical sequences are compared with each other. If the codeword assigned to a typical sequence is unique in the typical set (no other typical sequence has a codeword exactly the same as this one) bits number $l + 1$ and higher are not generated for this codeword, i.e. the encoding procedure for the related typical sequence (and all atypical sequences sharing the same codeword) is terminated. The iteration continues until the codewords generated for any pair of typical message sequences are different from each other.

The above explained encoding algorithm can be regarded as a tree structured random binning scheme with $m = 1$ such that when a bin is split to two smaller bins, one of these bins is selected by probability p and the other is selected by probability $1 - p$. In other words message sequences go to one bin with probability p and to the other one with probability $1 - p$.

Now we find the distribution of codeword length using this procedure. For this, let

us begin by finding the cumulative distribution function of codeword length, i.e. given a typical sequence x we are looking for $Pr(l_x \leq l)$, where l_x is the length of codeword assigned to x. Assume that codewords of length l bits are assigned to typical sequences. Let y be the codeword assigned to typical sequence x. The probability that y is unique in the typical set, i.e. no other typical sequence is assigned a codeword identical to y, is:

$$Pr(l_x \leq l) = \sum_{\mathbf{y}} Pr(\mathbf{y})(1 - Pr(\mathbf{y}))^{N_0}, \tag{3.34}$$

i.e. it is the probability that y is one of 2^l possible codewords and none of other N_0 typical sequences is assigned y. Since the parities are biased with $Pr(1) = p$, the probability of observing a sequence y with weight w is $p^w(1-p)^{l-w}$ and (3.34) can be rewritten as:

$$Pr(l_x \leq l) = \sum_{w=0}^{l} \binom{l}{w} p^w(1-p)^{l-w}(1 - p^w(1-p)^{l-w})^{N_0} \tag{3.35}$$

From this cumulative distribution, the length distribution of codewords can be found as $Pr(l_x = l) = Pr(l_x \leq l) - Pr(l_x \leq l-1)$, which is:

$$\begin{aligned} q_1(l) = &\textstyle\sum_{w=0}^{l} \binom{l}{w} p^w(1-p)^{l-w}(1 - p^w(1-p)^{l-w})^{N_0} \\ &- \textstyle\sum_{w=0}^{l-1} \binom{l-1}{w} p^w(1-p)^{l-w-1}(1 - p^w(1-p)^{l-w-1})^{N_0} \end{aligned} \tag{3.36}$$

Replacing p by $\frac{1}{2}$ in (3.36) gives $q_1(l) = (1-2^{-l})^{N_0} - (1-2^{-l+1})^{N_0}$, which is the same result as achieved for tree structured random binning with unbiased parities (See (3.12)).

Fig. 3.9 shows the probability mass function of the codeword length for a tree structured random binning scheme where parities are biased with $p = 0.2$. The number of typical sequences is $N_0 = 2^{40}$. From Fig. 3.9 the expected value of the codeword length is $\bar{l} = 57.67$. Given for comparison, the value of $log\, N_0/H(X)$ is 55.41 bits.

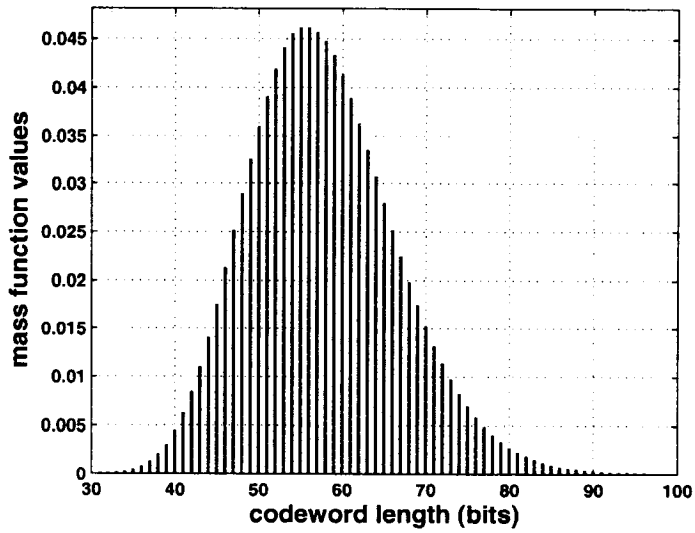

Figure 3.9: Probability mass function of the codeword length for a library of random binning schemes with $N_0 = 2^{40}$. The parities are biased with $Pr(1) = p = 0.2$. $\bar{l} = 57.67$ bits.

3.5 Conclusion

We studied the theoretical performance of nested error correcting codes for data compression. We presented a tree structured random binning method which is an iterative encoding method. In the proposed method parities are added to codewords until all typical sequences are distinguished from each other. We proved that tree structured random binning asymptotically achieves the entropy and derived the distribution of compression rate.

Chapter 4

Lossless Turbo Source Coding

In this chapter we consider the performance of short block length lossless turbo source coding. We optimize different components of the encoder to achieve improved compression rates. We introduce a new puncturing scheme with an improved performance at compression rates close to 0.5. Also, instead of square shape puncturing arrays, we use a rectangular puncturing array to allow for finer puncturing. Finally, we replace a single code with several codes operating in parallel to achieve better compression rates.

4.1 Introduction

For practical implementations of tree structured random binning, one may use an error correcting code (e.g. turbo [22] or RA [31] code). Let us call the encoder and the decoder of the related error correcting code as channel encoder and channel decoder, respectively. Then the source coding scheme works as follows

Source Encoder: The source encoder consists of a channel encoder and a channel decoder. The block diagram of this source encoder is depicted in Fig. 4.1. The source encoding procedure is an iterative encoding procedure, in which an incremental redundancy step is performed and then followed by a tentative decoding step.

(i) Incrementing redundancy: In the incremental redundancy step, the message is

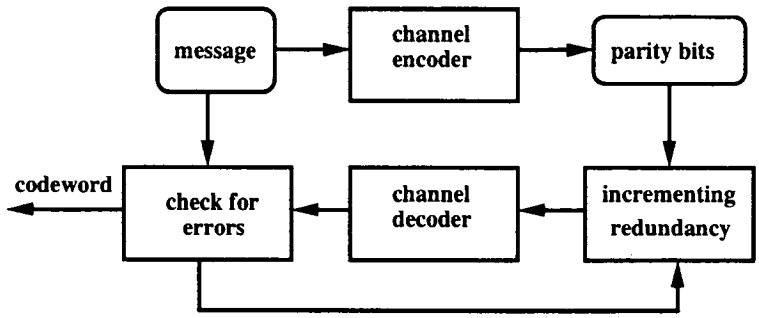

Figure 4.1: The block diagram of source encoder.

encoded using the channel encoder and the output sequence is stored (in case of systematic channel encoder, as in Fig. 4.1, the message bits are discarded and only the parities are stored).

(ii) Tentative decoding: In the tentative decoding step, for some integer $m \geq 1$, the first m bits of this output are used to decode the message[1], after which the decoder output is compared with the original message. If the message can be decoded with no error (i.e., the m bits are sufficient to describe the message), these m bits are transmitted to the decoder as the codeword. As long as no physical channel is involved, the decoder can decode the message with no error using this codeword.

(iii) Iteration: If the encoder observes errors after the tentative decoding, it appends m more bits to the codeword and runs the tentative decoding process again. The tentative decoding and incremental redundancy steps continue until the message can be recovered with no error.

Source Decoder: The source decoding is the same as the channel decoding. Furthermore if the source statistics are known at the decoder, they are used as *a priori* knowledge to achieve better performance. In this case, these statistics will also be used in the tentative decoding process at the encoder side.

When a physical channel is involved, the encoder could combat the channel noise by adding some extra parities to the generated codeword. The number of these parities

[1]In practice the encoder begins by assigning a pre-defined number of bits to the codeword, e.g. the closest number to $nH(X)$. This helps to decrease the number of decoding iterations [31].

could be defined based on the channel capacity and the error correcting capability of the underlying channel code. Also if the channel state information is available at the encoder, the encoder can take a more precise approach by passing the codeword through a simulated channel prior to each round of tentative decoding. One may notice that, in general, the codeword length is variable which may lead to error propagation [3]. However, the length of each codeword could be expressed by a few bits. These bits are highly protected to ensure error-free transmission, and are transmitted to the decoder (in other words these bits are transmitted via a *noiseless channel*). Therefore, the error propagation is avoided.

In what follows, we consider the lossless turbo source coding method, originally presented in [22]. For simplicity, we only focus on cases in which no physical channel is involved. We first review previous works on turbo source coding. We observe that lossless turbo source coding offers promising compression rates at the expense of having a large latency. To reduce this latency we propose a short block length turbo source coding scheme that achieves compression rates close to the entropy. For this, we modify different components of the original turbo source encoder [22]. Then we present numerical results to illustrate the effect of these modifications on the achieved compression rate. We also evaluate the computational complexity of the proposed encoding scheme.

4.2 Previous Work

Turbo codes were first introduced in [33]. Although these codes were originally designed for forward error correction (channel coding), they can also be used for data compression (source coding). Most of previous works on turbo source coding have focused on near-lossless coding where small levels of distortion can be tolerated (e.g., [12]-[21] and references therein). In general, near-lossless coding suffers from a drawback that renders its use in some important applications. That is, allowing a small level of average distortion results in occasional high-level distortions in some samples which can damage important

details of the information. This, of course, is not acceptable in applications such as compressing medical images where image details are of extreme importance [34]. Therefore, a completely lossless (we simply call it lossless) source code is required. Such a source code is also referred to as a zero-error code [35].

The original idea of lossless turbo source coding was introduced in [22] to compress binary biased i.i.d. sources. The block diagram of the turbo source encoder is shown in Fig. 4.2. In this case the turbo encoder consists of two parallel concatenated convolutional encoders and a code interleaver. A message of block length n bits is encoded by the two parallel concatenated convolutional codes. The two parity sequences, $\mathbf{y}_1$ and $\mathbf{y}_2$, are interleaved using a parity interleaver prior to storage. Let the source alphabet be $\chi = \{-1, +1\}$ (for simplicity of notation), $p = Pr(x = +1)$, and m be an even number that divides $2n$. At each encoding iteration, $m/2$ parities of each sequence are added to the codewords.

Notice that in the original work of [22], instead of adding parities and checking for the first successful tentative decoding, the procedure begins by considering all parities intact. Then parities are punctured step by step until tentative decoding fails. In other words a decremental redundancy approach is taken. In the sequel we use terms *puncturing* and *adding* parities alternatively toward the same goal which is lossless compression of the message. After finalizing iterative encoding the codeword is delivered to the channel decoder (for tentative decoding), along with the source *a priori* knowledge which is presented as an L-value, i.e. $log\frac{p}{1-p}$.

After performing the encoding process (i.e. reaching the first successful tentative decoding), the codeword, $\mathbf{y}$, and the codeword length, l, are multiplexed as a pair $(\mathbf{y}, l)$ and are sent to the decoder. Note that any sequence that is not compressed can be transmitted uncompressed along with a codeword length $l = n$. In this case, transmission of a flag bit is not necessary since the length $l = n$ reflects an uncompressed sequence.

The block diagram of a turbo source decoder is shown in Fig. 4.3. In Fig. 4.3, $\mathbf{L}$ is the L-value representing the source *a priori* knowledge, i.e. $\mathbf{L} = log\frac{p}{1-p}$. $\mathbf{Z}_j$, $\mathbf{A}_j$, $\mathbf{E}_j$,

and $\mathbf{D}_j$, $j = 1, 2$, are also L-values. $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are the L-values related to $\mathbf{y}_1$ and $\mathbf{y}_2$, respectively (after iterative encoding).$\mathbf{Z}_1$ and $\mathbf{Z}_2$ is zero if its related parity bit is erased, is ∞ if its related bit is intact and it is 1, and is ∞ if its related bit is intact and it is 1.$\mathbf{A}_1$ represents the *a priori* L-values for the first decoder. The first constituent decoder takes $\mathbf{L}$, $\mathbf{P}_1$ and $\mathbf{A}_1$ and calculates soft values $\mathbf{D}_1$ using the BCJR algorithm [22] (notice that $\mathbf{A}_1$ is not available in the first iteration). Then extrinsic information on the systematic bits, $\mathbf{E}_1$, is expressed as:

$$\mathbf{E}_1 = \mathbf{D}_1 - \mathbf{A}_1 - \mathbf{L}, \tag{4.1}$$

$\mathbf{E}_1$ is then interleaved and passed to the second decoder as the *a priori* input $\mathbf{A}_2$. The second decoder takes $\mathbf{L}$, $\mathbf{P}_2$ and $\mathbf{A}_2$, runs the BCJR algorithm to calculate $\mathbf{D}_2$ and feeds back extrinsic information $\mathbf{E}_2 = \mathbf{D}_2 - \mathbf{A}_2 - \mathbf{L}$ which becomes the *a priori* knowledge of the first decoder. This process iterates until convergence is achieved or a maximum number of iterations is reached (see [32] for details on turbo decoding algorithms).

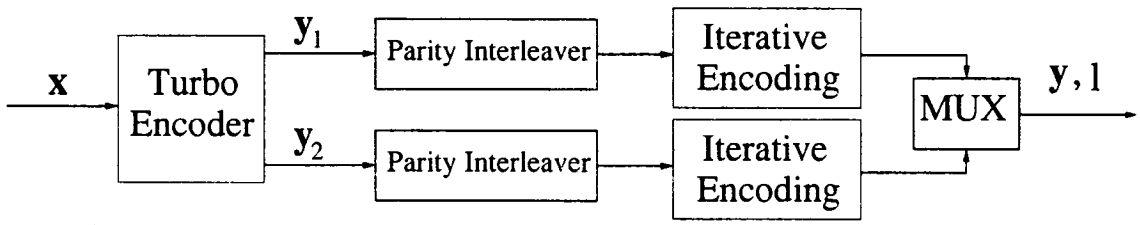

Figure 4.2: Block diagram of the turbo source encoder.

In Fig. 4.4, we show the histogram of the achieved compression rates for a turbo source coder with $n = 100$, and $m = 4$. The source is binary memoryless with $p = 0.10$ ($H(X) = 0.469$ bps). The turbo encoder consists of two identical recursive convolutional encoders with generator polynomial $G(D) = \frac{1+D^2}{1+D+D^2}$. The code interleaver and the parity interleaver are pseudo-random interleavers, i.e. they are randomly generated but are not altered after creation. From Fig. 4.4, the average compression rate of the turbo source encoder is found to be 0.667 bps. On the other hand, from (3.29), the average compression rate achieved by tree structured random binning with the same parameters is

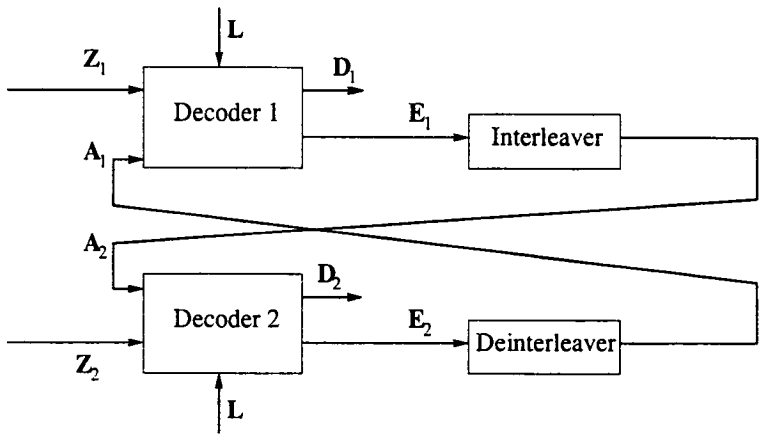

Figure 4.3: Block diagram of the turbo source decoder

equal $\widetilde{R}_b = 0.575$ bps.

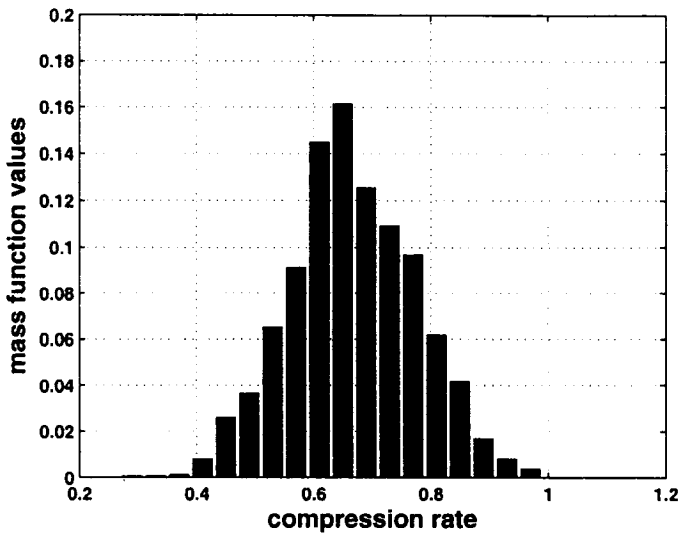

Figure 4.4: Probability mass function of the compression rate for a lossless turbo source encoder with $n = 100$, $p = 0.10$, and $m = 4$.

To achieve improved compression rates in lossless turbo source coding, several encoding/decoding iterations of large block lengths are required which results in a large latency [36]. Therefore, design of short block length schemes that achieve compression rates close to the entropy is of great interest. In this chapter, we propose a lossless compression scheme based on short block length turbo source codes. In this, we focus on

the design of the parity interleaver for different compression rates where we introduce a new puncturing technique that is shown to offer better compression rates than the ones introduced in the literature for a range of binary sources (i.e., entropy close to 0.5). For further enhancement, we employ (i) a finer puncturing method using rectangular arrays and (ii) a library of codes as opposed to a single code [22].

4.3 Proposed Encoding Scheme

In this section, we modify different components of the conventional turbo source encoder in [22] to achieve better compression rates for short block length systems.

4.3.1 System Model

Figure 4.5 shows a block diagram of the proposed turbo source encoder. The message block $\mathbf{u}$ is generated from an *i.i.d.* binary source with $Pr(1) = p$. As shown, a library of M turbo codes is used. Each turbo encoder block, in Fig. 4.5, consists of two constituent convolutional codes and one code interleaver. We consider the same constituent codes for all encoders but allow the code interleaver to change from one code to another. The turbo encoder number i produces two sets of parity bits, $\mathbf{y}_{i,1}$ and $\mathbf{y}_{i,2}$, which are interleaved using a parity interleaver Π_i. These parities are then written into two N_h by N_v arrays where $N_h \times N_v = n$ and n is the length of the message block, $\mathbf{u}$. The iterative puncturing process starts by puncturing the parities in one column of the array at each step, followed by checking the decodability of the punctured block before proceeding to the next step (this means puncturing $m = 2N_h$ parities at each encoding iteration). This puncturing process continues as long as lossless recovery of the message is still possible. All non-punctured parities as well as the number of punctured columns, S_i, are multiplexed as a pair $(\mathbf{y}_i, S_i)$. Among the outputs of all encoders, the minimum length output, say output number i^*, is chosen and $\mathbf{y}_{i^*}$ along with (S_{i^*}, i^*) are sent to the decoder. Note that transmitting S_{i^*} and i^* require $\lceil log_2(N_v/2) \rceil$ and $\lceil log_2 M \rceil$ excess bits, respectively, where

$\lceil log_2 M \rceil$ represents the smallest integer greater than or equal to $log_2 M$. Also note that the message bits are not sent since their statistics are known at the decoder. The decoding process is the same as the standard decoding of turbo codes [22]. In the original scheme of turbo source coding [22], $M = 1$, $N_h = N_v = \sqrt{n}$, and Π_1 is a random interleaver. It is shown in [22] that for a binary *i.i.d* source with entropy of 0.47 bps, one can achieve a compression rate of 0.60 bps using $n = 10000$ bits. Lempel-Ziv coding with the same block length achieves a compression rate of 0.66 bps [22].

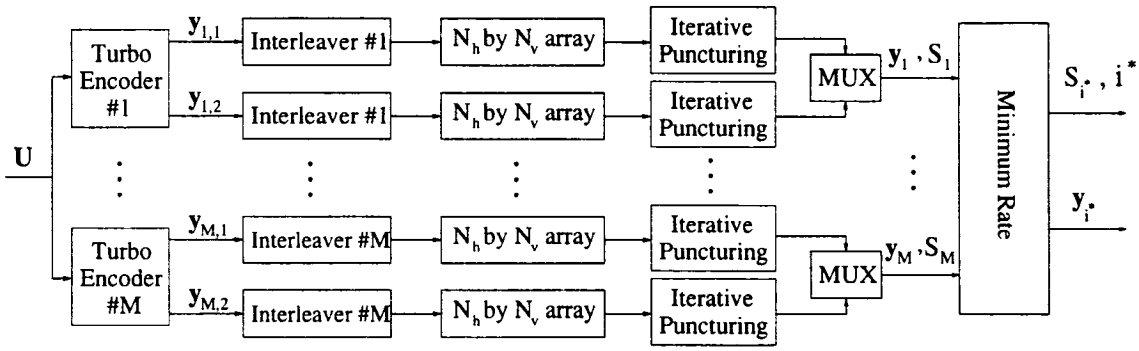

Figure 4.5: Block diagram of the proposed turbo source encoder.

4.3.2 Design of the parity interleaver

In [22] the parity interleaver is random, but it is not altered after being constructed. This leads to a pseudo-random puncturing of parities. In [27], the performance of pseudo-random puncturing is compared to that of a *structured* puncturing where the non-punctured parities have equal space on the trellis. It is shown in [27] that for compression rates close to one (i.e., sources with entropy close to one), pseudo-random puncturing performs better than *structured* puncturing. For smaller compression rates, on the other hand, *structured* puncturing performs better than pseudo-random puncturing. Here we consider the performance of these two puncturing methods along with a proposed puncturing method. The proposed puncturing scheme is shown to offer a performance close to *structured* puncturing at small compression rates, but with an improved performance at compression rates

close to one. For compression rates close to 0.5 (entropy close to 0.5), our proposed puncturing scheme outperforms both *structured* and pseudo-random puncturing. The concepts behind the proposed puncturing scheme arise from the design criteria of both *structured* and pseudo-random puncturing. In what follows we explain the operation of the *structured* and the proposed puncturing schemes.

If we let K be the number of non-punctured parities, *structured* puncturing works as follows:

(i) *Initialization*: Choose $l = \frac{n}{K}$.

(ii) For $j = 1$ to K: transmit parity bit number $\lfloor jl \rfloor$, where $\lfloor jl \rfloor$ is the largest integer less than or equal to jl.

Note that in *structured* puncturing, no parity interleaver is involved and the location of punctured parities differ from one rate to another.

Now we combine the ideas of pseudo-random puncturing and *structured* puncturing to design an interleaver that performs well, for a wider range of compression rates. Recall that in channel coding problems, the message bits are unbiased (each bit is either zero or one with probability 0.5) and hence, all parity bits are unbiased. On the other hand, in the dual source coding problem the message bits are biased and hence the parities will also be biased. In other words, parity bit number j, $1 \leq j \leq n$, can be modeled as a binary random variable with $Pr(1) = q_j$. Let $g_1(D)$ and $g_2(D)$ represent the feed-forward and feed-back polynomials of the convolutional encoder. It is proved in [37] that if and only if $g_1(D)$ is not divisible by $g_2(D)$ in GF(2), the encoder output is asymptotically uniform, i.e. as j increases, q_j goes to 0.5. But even in this case, still a few first parities might be considerably biased. For example, for a recursive convolutional encoder with feed-forward and feed-back polynomials $g_1(D) = 1 + D^2$, and $g_2(D) = 1 + D + D^2$, and a source with $p = .01$, simulations show that the first 110 parities have $q_j \leq .40$ (see Fig. 4.6). Based on this observation, the proposed puncturing method works as follows. The interleaver begins by choosing $K = \left\lfloor n\frac{H(X)}{2} \right\rfloor$ parities using the same algorithm explained for *structured* puncturing. These parities are interleaved and then written at the end of the

puncturing array. Usually all message blocks are compressed at rates above the entropy limit, and hence these parities are never punctured. However, when the block length is very small, some blocks are compressed to rates less than the entropy and some of these parities are punctured. This *structured* part of the interleaver maintains the maximum distance of non-punctured parities below a defined level and avoids long erasure bursts. Letting $f_j = \frac{1-q_j}{q_j}$, the remaining parities are interleaved using a pseudo-random interleaver, which draws the jth parity with probability $\frac{f_j}{\sum_{i=1}^{n} f_i}$. After drawing each parity, its related probability is forced to zero and the probabilities related to the remaining parities are rescaled. Using this method, most of the biased parities are placed at the beginning of the puncturing array. This assists in puncturing more biased parities and hence, achieving a more balanced compressed stream (the non-punctured parities are zero or one with probabilities closer to 0.5).

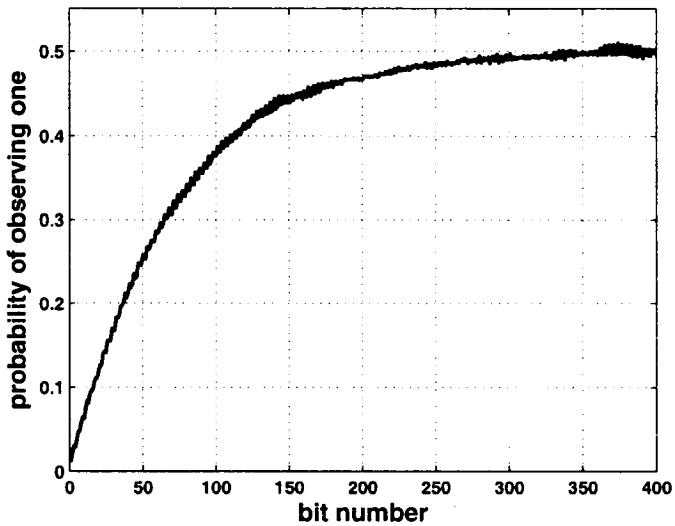

Figure 4.6: Probability of observing "one" in the output bit stream of a convolutional code with feed-forward and feed-back polynomials $g_1(D) = 1+D^2$, and $g_2(D) = 1+D+D^2$. The input stream is binary i.i.d. with $Pr(1) = .01$.

4.3.3 Rectangular puncturing arrays

In [22], a square parity puncturing array was considered ($N_h = N_c = \sqrt{n}$). In this case, the scheme allows to decrement the compression rate in steps of size $2/N_c$. Note that this step size may not be fine enough for sources with small entropy. Also as the block length decreases, the step size becomes larger. To achieve a finer puncturing, the square puncturing array can be replaced by an $N_h \times N_v$ rectangular array where $N_h < N_c$. This allows to decrement the compression rate in steps of size $2/N_v$, which improves the compression rate. This can be explained as follows. If we let N_h divide N_c, and by noticing $N_h N_v = N_c^2 = n$, the gain in compression rate can be bounded by:

$$-r_o \leq \overline{R}_s - \overline{R}_r \leq \frac{2}{N_c} - \frac{2}{N_v} - r_o, \tag{4.2}$$

where $\overline{R}_s$ and $\overline{R}_r$ are the average compression rates using square and rectangular puncturing arrays respectively, and r_o is the increase in compression rate due to the larger overhead size introduced by the rectangular array, given by

$$r_o = \frac{\lceil log(N_v/2) \rceil - \lceil log(N_c/2) \rceil}{n}. \tag{4.3}$$

4.3.4 Using a library of codes

The idea of using a library of codes was first proposed in [28] in the context of source coding using low density parity check (LDPC) codes. Similarly, for turbo source coding, we can employ M codes operating in parallel (see Fig. 4.5) and send the best result along with the number of punctured parities and the code index to the decoder. In this case, the average compression rate of the system is less than or equal to the average compression rate achieved by any of its codes.

4.4 Numerical Results

In the following results, we consider the constituent codes as recursive convolutional codes with a generator polynomial $g(D) = \frac{1+D^2}{1+D+D^2}$. Figure 4.7 shows the average compression rate of a lossless turbo source coder using the puncturing methods discussed in the previous section, for $n = 1024$ bits, $N_h = 32$, and $p = 0.01, 0.02, ..., 0.20$. As shown for $p \leq 0.09$, *structured* puncturing performs better than both the pseudo-random and the proposed puncturing schemes. The difference in compression rates is considerable for small rates. For instance, when $p = 0.01$ $(H(X) = 0.081)$, *structured* puncturing achieves 0.195 bps, while the pseudo-random and the proposed puncturing schemes achieve 0.266 bps, and 0.255 bps, respectively. For $p = 0.10 - 0.15$, the proposed puncturing scheme outperforms the other two schemes. For instance, when $p = 0.12$ $(H(X) = 0.529)$, our proposed puncturing scheme achieves 0.665 bps, while pseudo-random and *structured* puncturing achieve 0.704 bps, and 0.682 bps, respectively. For $p \geq 0.16$, pseudo-random puncturing achieves the lowest compression rate. This is shown, for example, for $p = 0.20$ $(H(X) = 0.722)$ where pseudo-random puncturing achieves 0.871 bps, while our proposed and *structured* puncturing schemes achieve 0.901 bps, and 0.921 bps, respectively. From these results one can conclude that for sources with entropies close to, 0, 0.5 and 1, *structured*, proposed and pseudo-random puncturing offer the best compression rates, respectively.

Figure 4.8 shows the histogram of the achieved compression rates for $n = 1024$ bits, when $N_h = 32$ and $N_h = 8$. The source is binary i.i.d. with $p = 0.05$ $(H(X) = 0.286$ bps) and a *structured* puncturing is used (offers the best compression rate for $p \leq 0.09$). The average compression rates in this case are 0.417 bps for $N_h = 32$, and 0.395 bps for $N_h = 8$. Note that the decrease in compression rate is achieved at the expense of increased encoding complexity. This is clear since with finer puncturing, more decodability tests and hence more decoding iterations are required to encode each message block.

In what follows, we evaluate the computational complexity of lossless turbo source coding.

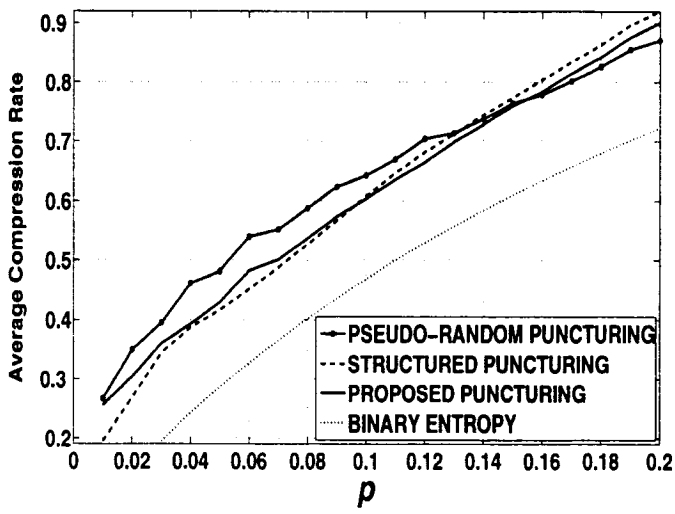

Figure 4.7: Average compression rate for $n = 1024$, $N_h = 32$, and different puncturing schemes.

In [38], the total number of additions and multiplications required for decoding one block of data are computed as follows[2]

$$A = 2 \times (6T - 4) \times I \times n, \tag{4.4}$$

$$B = 2 \times (10T + 2) \times I \times n, \tag{4.5}$$

where T is the number of states in the trellis diagram of each constituent code, I is the number of decoding iterations, A is the total number of additions, and B is the total number of multiplications. To compute the encoding complexity of turbo source coding, we replace the number of decoding iterations in (4.4) and (4.5) by the *average number of encoding iterations*. By the *average number of encoding iterations*, we mean the average number of puncturing steps required to encode each block multiplied by the maximum number of decoding iterations. Now we show how to evaluate the *average number of encoding iterations*. Let us denote the most probable compression rate by R_0. For example in Fig. 4.8.b, R_0 is the closest compression rate to 0.4 bps. Let the encoder begin with puncturing the proper number of parities to obtain R_0. If the punctured block is decodable,

[2]When the constituent codes are different, the coefficient 2 is replaced by a summation over different constituent codes [38].

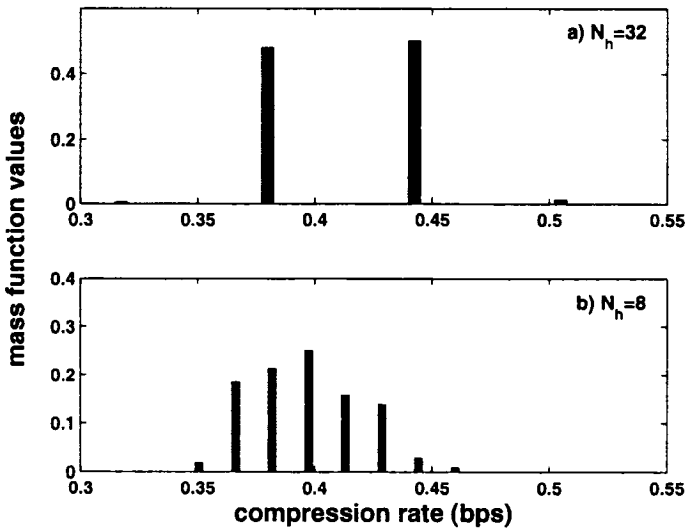

Figure 4.8: Probability mass function of compression rate for block length $n = 1024$ bits (a) $N_h = 32$, (b) $N_h = 8$. The source is binary i.i.d. with $p = 0.05$ ($H(X) = 0.286$ bps).

the puncturing process continues. But if the punctured block is not decodable, parities are added step by step until the block is decodable. For blocks that are encoded to rate R_0, two puncturing steps are required; one puncturing step is performed to confirm that R_0 is achievable, and the second is performed to ensure that no better rate can be achieved. Now define $\Delta R = 2/N_v$, and let $h > 0$ be an integer. For blocks that are encoded to rate $R_0 - h\Delta R$, $h + 2$ puncturing steps are required. In this case, the system requires $h + 1$ puncturing steps to confirm that this rate is achievable, and one more puncturing step is required to ensure that no better rate can be achieved. For blocks that are encoded to rate $R_0 + h\Delta R$, $h + 1$ puncturing steps are required. In this case, as soon as the system ensures that this rate is achievable, no more puncturing is required. This is simply because rates lower than $R_0 + h\Delta R$ were examined before. In order to evaluate the average number of puncturing steps required to encode each data block, we multiply the required number of puncturing steps that achieve each compression rate by its related probability and calculate the sum. Then, we multiply the obtained sum with the maximum number of decoding iterations to obtain the *average number of encoding iterations*. Notice that, choosing a setting point other than R_0 will result in a larger expected value of the encoding iterations.

Table 4.1 shows the average compression rate, $\overline{R}$, and the corresponding average number of encoding iterations, $\overline{I}$, versus p for a lossless turbo source encoder with a block length of 1024 bits and using either square or rectangular puncturing array (subscripts s and r, refer to square and rectangular puncturing respectively). For the rectangular puncturing array, the results are shown for the case of 8×128 array. Note that, these results are given for the best code design at the different compression rates (see Fig. 4.7). That is for, $p = 0.01, 0.05$, $p = 0.10$, and $p = 0.15, 0.20$, we use *structured*, proposed, and pseudo-random puncturing schemes respectively. It is observed that for $p = 0.01$, i.e. when the entropy is close to zero, the decrease in compression rate offered by the rectangular array is negligible (0.004 bps for $p = 0.01$). For $p = 0.05$, 0.10, 0.15, and 0.20, replacing the square array with a rectangular array results in 0.022 bps, 0.014 bps, 0.018 bps and 0.014 bps gain in the compression rate, and a corresponding 18%, 42%, 49% and 75% increase in the encoding complexity, respectively. Through simulations, we have noted that any further decrease in N_h results in a negligible reduction in the compression rate for all values of p. One may use (4.2) to prove that decreasing N_h from 8 to 4 will result in less than 0.007 bps reduction in compression rate. From (4.4) and (4.5), the number of additions and multiplications required to encode each data block are computed as $A = 40960 \times \overline{I}$, and $B = 86016 \times \overline{I}$, where $\overline{I}$ is replaced by $\overline{I}_s$ or $\overline{I}_r$. For example with $p = 0.05$, encoding a data block using a square puncturing array requires 1.01×10^6 additions and 2.12×10^6 multiplications. On the other hand, using a rectangular puncturing array with $N_h = 8$ will result in 1.19×10^6 additions and 2.50×10^6 multiplications.

So far, we have only considered the performance of the proposed code design for the case of a single code. In what follows we present simulation results for the case when more than one code is used (see Fig. 4.5). In Table 4.2, we show the average compression rate versus p, for $M = 1$ and $M = 4$ codes. These results are obtained for the case of square shape puncturing array, and based on the best puncturing method (i.e., according to Fig. 4.7 and similar to Table 4.1). One can see that using a library of 4 codes will result

Table 4.1: Average compression rate and the average number of encoding iterations using 32×32 square shape puncturing array ($\overline{R}_s$, $\overline{I}_s$) and 8×128 rectangular shape puncturing array ($\overline{R}_r$, $\overline{I}_r$).

p	$H(X)$	$\overline{R}_s$	$\overline{I}_s$	$\overline{R}_r$	$\overline{I}_r$
0.01	0.081	0.195	23.0	0.191	33.0
0.05	0.286	0.417	24.6	0.395	29.1
0.10	0.469	0.604	24.8	0.590	35.3
0.15	0.610	0.764	21.1	0.746	31.4
0.20	0.722	0.871	22.3	0.857	39.0

in a lower compression rate with a minimum of 0.015 bps for $p = 0.20$ up to 0.029 bps for $p = 0.05$. This will also increase the computational complexity M times, but since the codes are operating in parallel, no delay is added to the system. As a final study, Table 4.3 shows the achieved compression rates for the complete system described in Fig. 4.5 where we employ a rectangular array with $N_h = 8$ and a library of 4 codes for a data block of length $n = 1024$. Again the best puncturing method is used for each rate (i.e., according to Fig. 4.7 and similar to Table 4.1). Also shown in Table 4.3, for comparison, are the achieved compression rates $\overline{R}_o$ for the original turbo coding scheme in [22] with a block length of 10000 bits using a pseudo-random interleaver and a square puncturing array. It is observed that for $p = 0.20$, the scheme in [22] outperforms the proposed scheme. This is due to the larger block length used for the code presented in [22] (10000 bits, compared to 1024 bits used for the modified code).

Table 4.2: Average compression rate versus p, for $n = 1024$, using $M = 1$ code, and $M = 4$ codes.

p	0.01	0.05	0.10	0.15	0.20
$H(X)$	0.081	0.286	0.469	0.610	0.722
1 code	0.195	0.417	0.604	0.764	0.871
4 codes	0.167	0.398	0.591	0.742	0.856

Table 4.3: Average compression rate versus p, for $n = 1024$, $N_h = 8$, and $M = 4$ codes. Shown for comparison, $\overline{R}_o$ is the achieved compression rates for the original turbo coding scheme in [22] with a block length of 10000 bits using a pseudo-random interleaver and a square puncturing array.

p	0.01	0.05	0.10	0.15	0.20
$H(X)$	0.081	0.286	0.469	0.610	0.722
$\overline{R}_r$	0.165	0.386	0.575	0.721	0.835
$\overline{R}_o$	0.243	0.435	0.597	0.723	0.824

4.5 Conclusion

We considered the performance of short block length lossless turbo source coding. In this, we optimized different components of the encoder to achieve improved compression rates. We introduced a new puncturing scheme that was shown to offer a performance close to structured puncturing at low compression rates, but with an improved performance at compression rates close to one. For sources with entropy close to 0.5, our proposed puncturing scheme was shown to outperform both structured and pseudo-random puncturing methods introduced in the literature. Also, instead of square shape puncturing arrays, we used a rectangular puncturing array to allow for finer puncturing. Finally, we replaced a single code with several codes operating in parallel to achieve better compression rates.

Chapter 5

Detection of Code Index and Codeword Length in Turbo Source Coding

In this chapter we propose a detection algorithm to detect the codeword length and the code index at the decoder so that transmitting these numbers is no longer required. This will considerably reduce the compression rate for short block length systems. However, as the block length increases the effect of the proposed detection algorithm on the compression rate becomes less important.

5.1 Introduction

Consider the turbo source coding scheme proposed in Fig. 4.5. We observed that this scheme can achieve promising compression rates; however, the achieved rates are still far from the tree structured random binning bound. For example, from Fig. 4.4, the average compression rate of the turbo source encoder is found to be 0.667 bps. On the other hand, from (3.29), the average compression rate achieved by tree structured random binning with the same parameters is equal $\widetilde{R}_b = 0.575$ bps. A considerable part of the performance loss in short block length lossless turbo source coding is due to the transmission of the codeword length (S_{i^*} in Fig. 4.5) and index of the proper code (i^* in Fig. 4.5) to the

decoder. Transmitting S_{i^*} and i^* require $\lceil log_2(N_v/2) \rceil$ and $\lceil log_2 M \rceil$ excess bits, respectively, where $\lceil log_2 M \rceil$ represents the smallest integer greater than or equal to $log_2 M$. For example for $N = 100$, $N_v = 50$, and $M = 1$ (according to Fig. 4.4), 5 bits are required to transmit S_{i^*} which means having a 0.05 bps excess rate.

To reduce the compression rate for short block length systems, we propose a new detection algorithm. Using this algorithm, the decoder detects the codeword length and the code index from the transmitted codeword. Therefore their transmission is no longer required. In Section 5.2 we present the algorithm to detect the code index. Then, in Section 5.3 we modify the algorithm to detect the codeword length at the decoder.

5.2 Detecting the Code Index at the Decoder

Consider the turbo source decoder as shown in Fig. 4.3. To simplify the notations, let us show $\mathbf{A}_1$, the vector of *a priori* L-values of the first constituent decoder, by $\mathbf{A}$. Then, the (normalized) mutual information between $\mathbf{A}$ and the message,$\mathbf{x}$, is given by [22]:

$$I(\mathbf{x}; \mathbf{A}) = H(X) - \frac{1}{n} \sum_{k=1}^{n} log_2(1 + e^{-x_k L(x_k|a_k)}). \tag{5.1}$$

From the concept of the *extrinsic information transfer* (EXIT) chart [39] it is known that the sufficient condition to have successful decoding is that $I(\mathbf{x}; \mathbf{A})$ converges to $H(X)$, by increasing the number of decoding iterations. In (5.1), if for every k, $L(x_k|a_k) \rightarrow +\infty \times x_k$ then $I(\mathbf{x}; \mathbf{A}) \rightarrow H(X)$ and decoding is successful. When the matched decoder is used at the decoder, this event occurs with high probability. Note that $L(x_k|a_k) \rightarrow +\infty \times x_k$ implies that $a_k \rightarrow +\infty \times x_k$. Therefore, for the matched decoder case, the extrinsic L-values tend toward infinity by increasing the number of iterations. On the other hand, using a mismatched decoder is equivalent to using a decoder of a new code. Since the received parities do not necessarily belong to a codeword of this new code, the probability that the L-values tend toward infinity is low. Thus, a cost function $f(.)$ can

be defined to detect the proper code index using the *a priori* L-values. This function $f(.)$ should satisfy the following properties:

1- $f(.)$ should be symmetric about zero.

Because the reliability is defined based on the absolute values and the sign of L-values is not important in the decision.

2- $f(.)$ should be non-increasing between zero and infinity.

Because as the absolute value increases the reliability increases or at least does not decrease.

3- There is no penalty for infinite L-values, *i.e.*, $f(+\infty) = f(-\infty) = 0$.

There are different choices for such function. After testing many choices we found $f(x) = \exp(-|x|)$ to be a good choice, in the sense that it gives a lower detection error probability (see Subsection 5.2.1 for definition of detection error probability).

Using this function, detection of the code index is performed as follows. The received parities are decoded using all decoders. After finalizing the decoding process, the following parameter is computed for each code

$$\xi = \frac{1}{n}\sum_{k=1}^{n} \exp(-|a_k|), \tag{5.2}$$

and the code index is detected as the index of the decoder which achieves the minimum value of ξ. The decoded vector is then the output of this decoder. Usually, by increasing the number of decoding iterations, ξ converges to zero for the proper decoder. Figure 5.1 shows ξ versus the number of decoding iterations for $p = 0.02$, $n = 128$, $N_h = 4$ and $M = 2$ codes. The message has been encoded by code number 1. It is observed that for code number 1, the value of ξ reduces to zero, while for code number 2, ξ is more than 0.6.

5.2.1 Errors in Code Detection and Error-Free Detection

Two types of errors may occur in the process of detecting the code index.

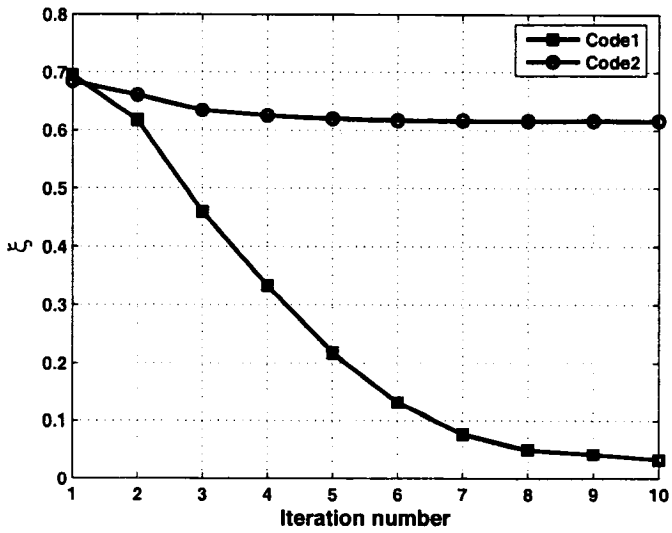

Figure 5.1: ξ versus the number of iterations for $n = 128$, $N_h = 4$, $M = 2$, and $p = 0.02$.

1- The first type of error occurs when the L-values do not tend to infinity (i.e., the mutual information does not converge to the entropy) even for the matched decoder case. Note that the tendency of the mutual information to the source entropy is a sufficient, but not necessary, condition for successful decoding [39]. In this case, ξ does not converge to zero for the correct code, and an error may occur.

2- The second type of error occurs when the received parities belong to a codeword of at least one other code, in addition to the proper code. In this case, ξ converges to zero for this code, and an error may occur.

It is possible that both types of errors occur simultaneously.

Let us denote the probability of detection error by P_d. When P_d is not zero, the compression is not completely lossless. To ensure that the compression is lossless, the encoder computes the value of ξ for each code and reassigns the code indexes such that $i = 1$ is assigned to the code with minimum value of ξ and $i = M$ is assigned to the code with maximum value of ξ. The distribution of this reassigned index is strictly non-uniform, thus it can be efficiently compressed using a Huffman code. For example the probabilities of reassigned indexes for a source with $p = .02$ and a turbo source encoder

with $n = 128$, $N_h = 4$, and $M = 4$ are 0.8732, 0.1015, 0.0210, and 0.0043. The average number of bits required to send this index by a Huffman code is 1.152 bits which is strictly less than 2 bits required to send the index without using the detection criterion and Huffman coding. If $\overline{R}$ represents the average compression rate when no Huffman code is used and $\overline{R}_d$ represents the average compression rate when the index is reassigned and transmitted, then:

$$\overline{R}_d = \overline{R} - \frac{\lceil log_2 M \rceil - \bar{l}_h}{n}, \tag{5.3}$$

where $\bar{l}_h$ is the average number of bits required to transmit the reassigned index by Huffman code (transmission of the code index without applying the detection criterion and reassignment requires $\lceil log_2 M \rceil$ bits). Thus, the gain achieved using the proposed detection algorithm can be expressed in bits per sample (bps) as follows:

$$r = \frac{\lceil log_2 M \rceil - \bar{l}_h}{n}. \tag{5.4}$$

5.2.2 Numerical Results

In this section, we provide some numerical results. For all simulations $N_h = 4$, convolutional codes are systematic with feed-back generator polynomial $g_0(D) = 1 + D + D^2$ and feed-forward polynomial $g_1(D) = 1 + D^2$. The parity interleaver is a block interleaver with 8 rows for $p = .02$ and $p = .05$, and 4 rows for $p = .08$. For each code, 10^4 message blocks are compressed and 10 iterations are performed for turbo decoding. Tables 5.1-5.3 show the gain achieved by the proposed detection algorithm, for different system parameters. In Table 5.1, $p = 0.02$, $n = 128$, and M, the number of codes, increases from 4 to 16. It is observed that the proposed method performs better for larger number of codes. In the case of $M = 16$ codes, our detection method helps to reduce the compression rate by 0.015 bps, which is more than 10% of the entropy ($H(X) = 0.141$ bps) and 7% of $\overline{R}$. Table 5.2 shows that for fixed M and p, by increasing the block length, the index detection algorithm becomes less effective. From Table 5.3 one can observe that when M and n

are fixed, by increasing p, r slightly increases. However, the detection algorithm is more effective for smaller values of p, where r contains a larger fraction of the compression rate.

Table 5.1: The gain of the index detection algorithm versus the number of codes, for $p = 0.02$ and $n = 128$ bits.

M	$\overline{R}$ (bps)	P_d	$\bar{l}_h$	$\overline{R}_d$ (bps)	r (bps)
4	0.250	0.127	1.152	0.243	0.007
8	0.245	0.345	1.591	0.234	0.011
16	0.243	0.429	2.104	0.228	0.015

Table 5.2: The gain of the index detection algorithm versus the code block length, for $p = 0.02$ and $M = 8$.

n	$\overline{R}$ (bps)	P_d	$\bar{l}_h$	$\overline{R}_d$ (bps)	r (bps)
64	0.295	0.557	2.095	0.281	0.014
128	0.245	0.345	1.591	0.234	0.011
256	0.236	0.098	1.148	0.229	0.007

Table 5.3: The gain of the index detection algorithm versus p, for $M = 8$ and $n = 128$ bits.

p	$\overline{R}$ (bps)	P_d	$\bar{l}_h$	$\overline{R}_d$ (bps)	r (bps)
0.02	0.245	0.345	1.591	0.234	0.011
0.05	0.408	0.073	1.084	0.393	0.015
0.08	0.539	0.023	1.024	0.524	0.015

5.3 Detection of Codeword Length at The Decoder

In this section we assume only one code is used and show how to detect the codeword length at the decoder. The proposed approach is similar to the approach introduced to detect the code index. Again, defining the soft value, ξ, as

$$\xi = \frac{1}{n}\sum_{k=1}^{n} exp(-|a_k|), \tag{5.5}$$

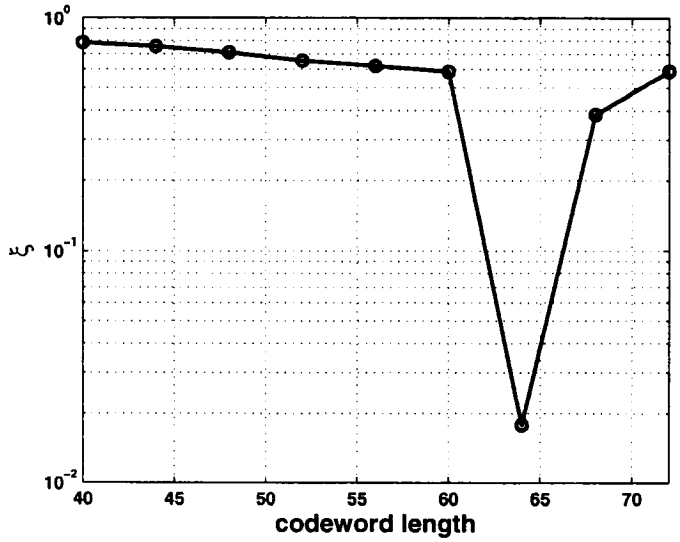

Figure 5.2: The value of ξ versus the codeword length, for $p = 0.10$, $n = 100$, $m = 4$, and 10 decoding iterations.

we show how the decoder can detect the codeword length (in bits) from the received bit stream. The decoder begins by assuming the codeword length $l = m = 2N_h$ ($S_{i^*} = 1$), i.e. it takes the first m received parities, performs a defined number of iterations and then computes the value of ξ from (5.5). Then l is increased to $2m$, and the decoding procedure is repeated for the first $2m$ parities. As long as all the parities belong to the same codeword, ξ will decrease by increasing l. This is true since obtaining more valid information increases the absolute values of a_i's and hence ξ decreases. Note that further increase of l leads to the case where the parities of the next codeword are appended to the parities of the current codeword. In this case, the decoder receives incorrect information and ξ increases. Thus, the proper value of l can be detected. Fig. 5.2 shows the value of ξ versus l for one message block. In Fig. 5.2, $p = 0.10$, $n = 100$, $m = 4$ ($N_h = 2$), and 10 decoding iterations are performed for both detection and decoding[1]. The actual value of l is 64 bits, which is shown to be correctly detected.

[1]Notice that the number of turbo decoding iterations performed for detection can be different from the number of iterations performed to decode the message.

5.3.1 Errors in Detection of Codeword Length and Error-Free Detection

Similar to the case of detecting the code index, the detection of codeword length is also not always successful. In some cases a detection error may occur, i.e. ξ reaches its minimum at length $l_d \neq l$. To study these error events, let us consider the pattern of errors in the example above (Fig. 5.2). Given a detection error, the expected values of l and l_d are found to be $\bar{l} = 69.34$ and $\bar{l}_d = 76.54$, respectively. Also, if we call the value of ξ at l_d by ξ_{min}, the average value of ξ_{min} for all message sequences is found equal to 0.033. However, computing this value exclusively for erroneous sequences shows an increase to 0.161. From this observation the occurrence of detection errors can be justified as follows. The convergence of ξ to zero is not a necessary condition for terminating the encoding process. The encoder has a built-in decoder so it decodes the codeword and compares its binary output with the message. As soon as this output is identical to the message, the encoding process is terminated. However, sometimes the value of ξ is still significant when the encoding process is terminated. In this case minimizing ξ at l cannot be guaranteed and detection error may occur. By noticing that $\bar{l} = 69.34$ is larger than $n \times \overline{R} = 61.7$ bits (without considering the 5 bit overhead), we found out that this error mainly occurs for high-weight atypical sequences (which have longer codewords). In fact, simulation results show that the expected weight of the message sequence when an error occurs is 13.01 which is larger than $n \times p = 10$. Figure 5.3 shows the value of ξ for a codeword of length 76 bits. As shown from these results, the detected codeword length is 80 bits.

Since errors may occur in the detection process, the encoding procedure is not lossless anymore. However, lossless encoding can be guaranteed by checking the detection criterion at the encoder and prior to transmission. In this, the encoder will always send a flag bit to show whether the detection procedure is successful or not. If the detection is not successful, the codeword length is transmitted with $\lceil log(n/m) \rceil$ ($\lceil log\, N_v/2) \rceil$) bits,

after the flag bit. Given this detection criterion, only $1 + \lceil log(n/m) \rceil P_d$ excess bits are transmitted where P_d represents the probability of detection error. If the number of these excess bits is less than $\lceil log(n/m) \rceil$, or equivalently as long as

$$P_d < 1 - \frac{1}{\lceil log_2(n/m) \rceil}, \tag{5.6}$$

the proposed detection criterion will assist in reducing the compression rate of the lossless turbo source coder. If we denote the average compression rate with and without our detection criterion, respectively, by $\overline{R}_d$ and $\overline{R}$, then

$$\overline{R}_d = \overline{R} - \frac{(1 - P_d) \lceil log_2(n/m) \rceil - 1}{n}. \tag{5.7}$$

For example if we consider a turbo source code with $n = 100$ and $m = 4$, and a source with $p = 0.10$, then the probability of detection error is $P_d = 0.027$ and the corresponding $\overline{R}_d = 0.628$ bps (without the detection method, $\overline{R} = 0.667$ bps). It is observed that the proposed detection method reduces the compression rate by 0.039 bps. Compared to the random binning bound, the compression rate $\overline{R}_d$ is 0.053 bps far from $\widetilde{R}_b$.

Notice that similar to the approach proposed for detecting the code index, the codeword length could be compressed by a Huffman code. However, simulations show that in this case the gain achieved by applying Huffman coding is negligible.

5.3.2 Numerical Results

To examine the performance of the proposed detection method under different encoding parameters, in Table 5.4 we show the value of P_d for two different turbo codes using different block lengths. The source is binary i.i.d. with $p = 0.10$. P_{d1} and P_{d2} represent the detection error probability for Code 1 and Code 2, respectively. In these results, Code 1 consists of two identical recursive convolutional codes with $G(D) = \frac{1+D^2}{1+D+D^2}$ where code 2 consists of two identical recursive convolutional codes with $G(D) = \frac{1+D^3+D^4}{1+D+D^2+D^4}$. It is observed that P_d depends on both the block length and the code structure. Also as

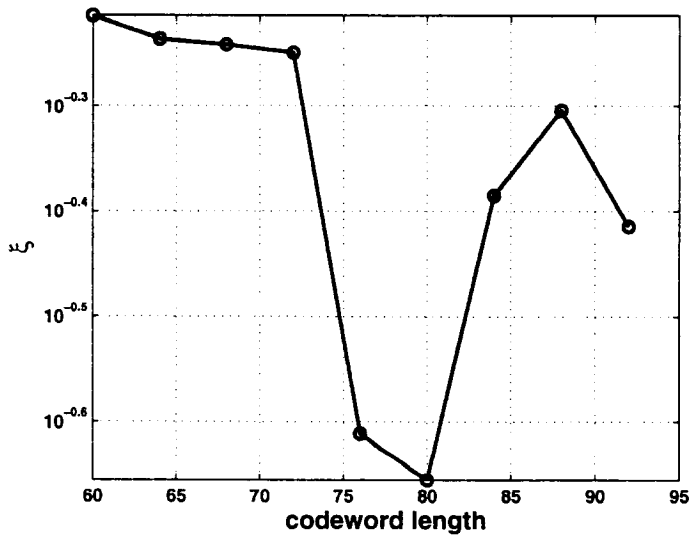

Figure 5.3: The value of ξ versus the codeword length, for $p = 0.10$, $n = 100$, $m = 4$, and 10 decoding iterations. The correct codeword length is 76 bits but the detected length is 80 bits.

seen, as the block length increases P_d decreases. The reduction of compression rate using the proposed detection method is also given in Table 5.4. In Table 5.4, $r_1 = \overline{R}_1 - \overline{R}_{d1}$ represents the reduction of compression rate for Code 1 and $r_2 = \overline{R}_2 - \overline{R}_{d2}$ represents the reduction of compression rate for Code 2. We noted that for a fixed block length, these values are almost the same for both codes. Also it is observed that as the block length increases, r_1 and r_2 decrease, i.e. the performance loss avoided using the proposed detection method is less significant for larger block lengths. In fact, since $P_d \geq 0$, one may note from (5.7) that for any given code, $\overline{R} - \overline{R}_d \leq \frac{\lceil log_2(n/m) \rceil - 1}{n}$. For example, for $m = 4$ and $n = 800$, $\overline{R} - \overline{R}_d \leq 0.009$, whereas for $m = 4$ and $n = 1600$, $\overline{R} - \overline{R}_d \leq 0.005$. Therefore, this method is not appropriate for large block length systems.

Up to this point, we have only considered the case where the number of decoding iterations used for detecting the number of transmitted parities is fixed to that used for decoding the message. In Fig. 5.4, we examine the performance of the proposed detection method using different number of decoding iterations. These results are shown for code 1, with a block length $n = 100$, $m = 4$ and a binary *i.i.d.* source with $p = 0.10$. As

Table 5.4: P_d and the reduction of compression rate for different block lengths and different code structures. $m = 4$, $p = 0.10$, and 10 decoding iterations are performed.

n	100	200	400
P_{d1}	$2.7e-2$	$4.6e-3$	$1.0e-3$
$\overline{R_1}(bps)$	0.667	0.652	0.625
r_1	0.039	0.025	0.015
P_{d2}	$3.2e-2$	$1.0e-2$	$2.8e-3$
$\overline{R_2}(bps)$	0.651	0.627	0.595
r_2	0.038	0.025	0.015

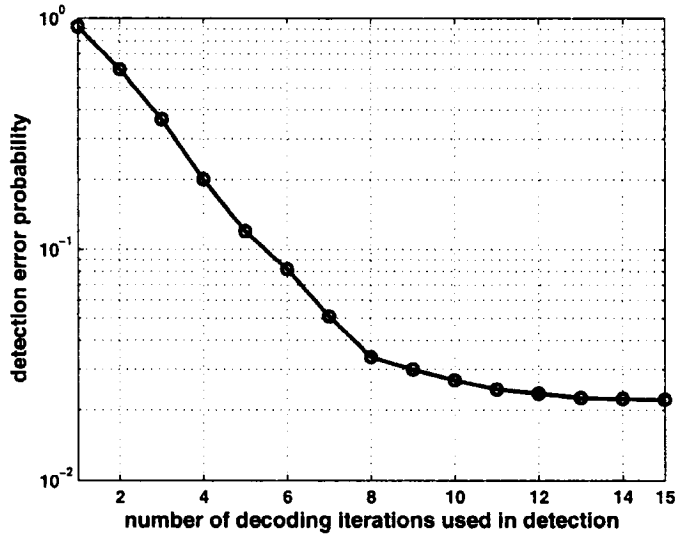

Figure 5.4: P_d versus the number of decoding iterations for $n = 100$, $m = 4$, and $p = 0.10$.

evident from these results, the probability of detection error is a function of the number of decoding iterations. Also, from Fig. 5.4, it is observed that P_d flattens after 14 iterations. Hence, by changing the number of iterations, one may trade-off P_d for the decoding complexity.

5.4 Conclusion

We observed that the achieved compression rate of lossless turbo source coding is far from the tree structured random binning bound. This difference in performance was shown to be partly due to the transmission of the code index and the codeword length, which requires a considerable excess rate for short block length codes. To improve the performance of turbo source coding, instead of sending the code index and the codeword length, we proposed methods to detect this information at the decoder side. The proposed detection methods were shown to improve the performance of the lossless turbo source coding. From the numerical results we conclude that the proposed detection algorithms are more effective for sources with small entropies and systems with short block lengths. As the number of codes increases, the algorithm achieves higher gain (in case of detecting the code index). Finding a better detection criterion leads to a more considerable gain and smaller compression rates for the system.

Chapter 6

Conclusion

We starts by summarizing the key points of each chapter. Our main contributions are presented in Section 2. Section 3 gives some suggestions for future research.

6.1 Summary

In this thesis, we considered the performance of nested error correcting codes for lossless data compression.

In Chapter 2 (background) we reviewed the concept of typical sets and presented entropy as the limit of lossless compression. We observed that each source X can be compressed without any loss of information as long as the compression rate is greater than the source entropy $H(X)$. Standard data compression techniques like Huffman coding and Lempel-Ziv coding are able to achieve the entropy limit. However, transmission of data over a noisy channel will cause errors and since standard compression codes are not capable of error correction, the whole output sequence will be corrupted. This fact drew attentions to the possibility of combining two fundamental problems of data transmission: data compression and error correction. In other words if one finds codes that are capable of performing both data compression (source coding) and error correction (channel coding), the source coding and channel coding blocks of a transmission system can be combined

as a joint source channel coding unit. Shannon was the first one to investigate the fact that channel codes themselves could be employed as good source codes (or vice versa). In recent decades Shannon's work was followed by other researchers and application of channel codes for data compression received increasing attention. As a first step toward designing good source codes, it was proved that random codes could achieve entropy. This proof was formulated in [3] and the proposed scheme was called random binning. The basic idea of random binning is simple; choose a compression rate above entropy, generate codewords using random parities for each sequence, transmit the codeword and use the optimal decoder at the receiver. Given these, it was proved that the probability of error over the typical set approaches zero as the block length approaches infinity. Theoretical success of random codes suggests that random-like codes can serve as practical source codes because they generate random-like codewords and have constructive near-optimum decoding schemes. Turbo codes, LDPC codes, RA codes, and fountain codes are examples of random-like codes which have been successful in achieving near-entropy compression rates. If we only care about the probability of error over the typical set, i.e. near-lossless compression is desired, the complexity of source coding is very low. The encoder only generates and transmits the codeword, as it does in random binning or in the conventional channel coding problem. However, due to different reasons, sometimes a completely lossless compression is desired. In this case the source encoder must have a built-in decoder and check for possible decoding errors. This process is called tentative decoding. If tentative decoding is not successful, the codeword length will be increased by adding more parities and tentative decoding is performed again. This is equivalent to using a nested code, i.e. a code where each codeword of a higher-rate code is formed by adding parities to a codeword of a lower-rate code. Iterative process of adding parities and checking for errors continues until successful tentative decoding is reached. This is why completely lossless codes have complicated source coding process which adds more delay to the system.

After presenting the background, we considered two important problems receiving

little attention in literature. The first problem was to establish a theory showing how the compression rate changes as the block length increases.

In Chapter 3 we considered completely lossless data compression using nested error correcting codes. Using nested codes implies that the codeword is variable-length and the compression rate is a random variable. Finding the probability mass function of this random variable was of interest. In Chapter 3 we established a tree structured random binning theory to show how nested random codes perform lossless compression. The probability mass function of the compression rate was derived and it was proved that tree structured random binning asymptotically achieves entropy. Furthermore, we considered a more practical problem of tree structured random binning using biased parities and derived the distribution of compression rate in this case.

The second important problem to consider was the role of delay in the system. Most communications systems have a maximum threshold for delay which cannot be exceeded. We considered completely lossless turbo source coding as an example of practical schemes and investigated satisfaction of the delay constraint. We observed that existing turbo source coding schemes achieve promising compression rates; however, their large block length (order of 10000 bits) causes a large delay. To reduce this delay we turned to short block length codes.

In Chapter 4 we designed short block length turbo source codes with low compression rates. To this end, we showed how to modify different components of the source encoder to reduce the compression rate. Specifically we investigated design of parity interleavers, applied rectangular puncturing arrays, and replaced a single turbo code by a library of turbo codes to decrease the compression rate. We observed that applying the proposed modifications dramatically reduces the compression rate. We also investigated computational complexity of encoding for original and proposed schemes.

In Chapter 5 we showed how to detect the overhead introduced by the code index and the codeword length at the decoder. Using the proposed detection technique, this overhead is no longer transmitted since it can be detected from the codeword. Therefore,

the compression rate is further decreased. We must mention that the detection technique becomes less attractive for large block length systems.

6.2 Contributions

The main contributions presented in this thesis can be summarized as follows:

- We considered the theoretical performance of nested error correcting codes for data compression. For this, we generalized the conventional random binning argument to a so called tree structured random binning concept. We derived the distribution of the compression rate achieved by the proposed tree structured random binning scheme. Comparing this distribution with the distribution achieved using a library of random binning schemes, we proved that a nested code can achieve rates close to a library of codes but with much lower encoding/decoding complexity.

- For large data blocks, the proposed scheme was shown to asymptotically achieve the entropy limit.

- As a practical implementation of this random binning scheme, we examined the performance of the lossless turbo source coding scheme proposed in [22]. Considering short block length codes, we modified different components of the encoder to achieve better compression rates.

- We presented a detection algorithm to eliminate the overhead introduced by transmission of the codeword length and the code index. Using such an iterative detection technique proved to further reduce the compression rate for short block length turbo source codes.

6.3 Future Work

For future research we propose generalizing the tree structured random binning theory for random-like codes. In other words, we propose to derive the distribution of the compression rate for turbo source codes and LDPC source codes. This could be performed in two steps. First, one analyzes the rate distribution by considering the actual code, e.g., turbo code or LDPC code, with the optimum decoder (maximum likelihood decoder). Then the distribution is analyzed by considering the practical decoding, e.g., BCJR in case of turbo codes or belief propagation in case of LDPC codes. This generalization will give us an insight how a practical scheme works and provides us with guidelines for designing better data compression schemes.

We also suggest generalizing the tree structured random binning theory for sources with memory in order to find out how the scheme achieves the entropy rate. Generalization of this theory to joint compression of correlated sources is another interesting problem.

Bibliography

[1] J. Ziv, A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, Vol. 23, Issue 3, May 1977, pp. 337-343.

[2] C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," *In IRE Nat. Conv. Rec.*, Mar. 1959, pp. 142–163.

[3] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, 1991.

[4] C.E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, July 1948, pp. 379-423.

[5] T. M. Cover, Mung Chiang, "Duality between channel capacity and rate distortion with two-sided state information," *IEEE Trans. Inform. Theory*, Vol. 48, Issue 6, June 2002, pp. 1629-1638.

[6] S. S. Pradhan, J. Chou, K. Ramachandran, "Duality between source coding and channel coding and its extension to the side information case," *IEEE Trans. Inform. Theory*, Vol. 49, Issue 5, May 2003, pp. 1181-1202.

[7] M. W. Marcellin, T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Trans. Commun.*, vol. 38, Jan 1990, pp. 82-93.

[8] M. V. Eyuboglu, G. D. Forney, "Lattice and trellis quantization with lattice and trellis-bounded codebooks-High rate theory for memoryless sources," IEEE Trans. Inform. Theory, vol. 39, Jan 1993, pp. 46-59.

[9] R. Laroia, N. Farvardin, S. A. Tretter, "On optimal shaping of multidimentional constellations," *IEEE Trans. Inform. Theory*, vol. 40, July 1994, pp. 1044-1056.

[10] R. Laroia, N. Farvardin, "A structured fixed-rate vector quantizer derived from a variable length scalar quantizer-Part I: Memoryless sources," *IEEE Trans. Inform. Theory*, vol. 39, May 1993, pp. 851-867.

[11] D. Slepian, J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inform. Theory*, vol. 19, Issue 4, July 1973, pp. 471-480.

[12] J. Garcia-Frias, and Y. Zhao, "Compression of correlated binary source using turbo codes," *IEEE Communications letters*, vol. 5, no. 10, Oct. 2001, pp. 417–419.

[13] J. Bajcsy, and P. Mitran, "Coding for the Slepian-Wolf problem with turbo codes," *In Proc. IEEE Globecom 2001,* Nov. 2001, pp. 1400–1404.

[14] A. Aaron, and B. Girod, "Compression with side information using turbo codes," *In Proc. Data Compression Conference, DCC 2002,* Apr. 2002, pp. 252–261.

[15] J. Garcia-Frias, and Wei Zhong, "LDPC codes for compression of multi-terminal sources with hidden Markov correlation," *IEEE Communications Letters*, vol. 7, no. 3, Mar. 2003, pp. 115–117.

[16] S. Yang, and P. Qiu, "On the performance of linear Slepian-Wolf codes for correlated stationary memoryless sources," *In Proc. Data Compression Conference, DCC 2005*, Mar. 2005, pp.53–62.

[17] A. D. Liveris, Z. Xiong, C. N. Georghiades, "Compression of binary sources with side information at the decoder using LDPC codes," *IEEE Communications Letters*, vol. 6, no. 10, Oct. 2002, pp. 440–442.

[18] T. Tian, J. Garcia-Frias, and W. Zhong, "Compression of correlated sources using LDPC codes," *In Proc. Data Compression Conference, DCC 2003,* March 2003.

[19] Y. Yang, V. Stankovic, Z. Xiong, and W. Zhao; "On multiterminal source code design," *In Proc. Data Compression Conference*, DCC 2005, Mar. 2005, pp. 43–52.

[20] A.D. Liveris, Z. Xiong, and C. N. Georghiades, "Joint source-channel coding of binary sources with side information at the decoder using IRA codes," *IEEE Workshop Multimedia Signal Processing*, 9-11 Dec. 2002, pp.53–56.

[21] J. Muramatsu, T. Uyematsu, and T. Wadayama, "Low-density parity-check matrices for coding of correlated sources," *IEEE Trans. Inform. Theory*, vol. 51, no. 10, Oct. 2005, pp. 3645–3654.

[22] J. Hagenauer, J. Barros, and A. Schaefer, "Lossless turbo source coding with decremental redundancy," *in Proc. 5th international ITG conference on Source and Channel Coding, SCC'04,* Erlangen, Germany, Jan. 2004.

[23] E. Weiss, "Compressing and coding," *IRE Trans. Inform. Theory*, Apr. 1962, pp. 256–257.

[24] P. E. Allard and A. W. Bridgewater, "A source coding technique using algebraic codes," *In Proc. 1972 Canadian computer conference*, June 1972, pp 201–213.

[25] K. C. Fung, S. Tavares, and J. M. Stein, "A comparison of data compression schemes using block codes," *In Proc. IEEE international Electrical and Electronics conference*, Oct. 1973, pp. 60–61.

[26] T. Ancheta, "Syndrome source coding and its universal generalization," *IEEE Trans. Inform. Theory*, vol. 22, no. 4, July 1976, pp. 432–436.

[27] J. Garcia-Frias, and Y. Zhao, "Compression of binary memoryless sources using punctured turbo codes," *IEEE Communications Letters*, vol. 6, no. 9, Sept. 2002, pp. 394–396.

[28] G. Caire, S. Shamai, and S. Verdu, "A new data compression algorithm for sources with memory based on error correcting codes," *in Proc. IEEE Information Theory workshop*, Paris, France, 31 Mar-4 Apr. 2003, pp. 291–295.

[29] G. Caire, S. Shamai, and S. Verdu, "Noiseless data compression with low-density parity-check codes," DIMACS series in discrete mathematics and theoretical computer science, vol. 66, pp. 263-284, American Mathematical Society, 2004 .

[30] G. Caire, S. Shamai, A. Shokrollahi, and S. Verdu, "Fountain codes for lossless data compression," DIMACS series in discrete mathematics and theoretical computer science, vol. 68, pp. 1-20, American Mathematical Society, 2006.

[31] N. Dütsch, "Code optimization for lossless turbo source coding," *In Proc. IST mobile & wireless communications summit*, Dresden, Germany, June 2005.

[32] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2., Mar. 1996, pp. 429–445.

[33] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: turbo codes," *In Proc. ICC 93*, pp. 1064-1070, May 1993.

[34] E. Tuncel, P. Koulgi, S. Regunathan, and K. Rose, "Zero-error Coding With Maximum Distortion Criterion," *In Proc. Data Compression Conference, DCC 2002*, April 2002.

[35] P. Koulgi, E. Tuncel, S. L. Regunathan, and K. Rose, "On zero-error coding of correlated sources," *IEEE Trans. Inform. Theory*, Vol. 49, No. 11, pp. 2856-2873, Nov. 2003.

[36] F. Daneshgaran, and M. Mondin, "Optimized turbo codes for delay constrained applications," *IEEE Trans. Inform. Theory*, Vol. 48, Issue 1, pp. 293-305, Jan. 2002.

[37] Guang-Chong Zhu, F. Alajaji, J. Bajcsy, and P. Mitran, "Transmission of nonuniform memoryless sources via nonsystematic turbo codes," *IEEE Trans. Commun.*, Vol. 52, No. 8, pp. 1344-1354, Aug. 2004.

[38] D. J. Costello Jr., A. Banerjee, C. He, and P. C. Massey, "A comparison of low complexity turbo-like codes," *In Proc. 36th asilomar conf. on signals, systems, and computers,* Pacifiv Grove, California, USA, Nov. 2002.

[39] S. Ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, Vol. 49, No. 10, pp. 1727-1737, Oct. 2001. Tuncel, P. Koulgi, S. Regunathan, and K. Rose, "Zero-error Coding With Maximum Distortion Criterion," *In Proc. Data Compression Conf., DCC 2002*, Apr. 2002.

[40] J. Haghighat, W. Hamouda, M. R. Soleymani, "Lossless source coding using nested error correcting codes," To Appear IEEE Trans. Signal Processing, Accepted Aug. 2006.

[41] J. Haghighat, W. Hamouda, M. R. Soleymani, "Design of lossless turbo source encoders," IEEE Signal Processing Letters, vol. 13, Issue 8, Aug. 2006, pp. 453-456.

[42] J. Haghighat, M. R. Soleymani, W. Hamouda, "Code detection in turbo source coding," IEEE Communications Letters, vol. 10. Issue 4, April 2006, pp. 225-227.

[43] J. Haghighat, W. Hamouda, M. R. Soleymani, "Lossless source coding using tree structured random binning," In Proc. ISIT 2006, Seattle, WA, July 2006, pp. 2348-2352.

[44] J. Haghighat, W. Hamouda, M. R. Soleymani, "Fixed-to-variable length source coding using turbo codes," IEEE 2006 workshop on signal processing systems (SIPS 2006), Banff, AB, Canada, Oct. 2006.

[45] J. Haghighat, W. Hamouda, M. R. Soleymani, "Random binning and turbo source coding for lossless compression of memoryless sources," In. Proc. VTC fall 2006, Montreal, QC, Canada, Sept. 2006.

[46] J. Haghighat, M. R. Soleymani, W. Hamouda, "Detection of code index in turbo source coding," In. Proc. IEEE Globecom 2005, St. Louis, Missouri, 28 Nov.-2 Dec. 2005, pp. 1397-1401.

[47] J. Haghighat, M. R. Soleymani, W. Hamouda, "Design of short block length lossless turbo source encoders," In. Proc. VTC fall 2005, Dallas, Texas, Sept. 2005, pp. 527-531.