# Trapping Sets in Fountain Codes over Noisy Channels

by

Vivian Lucia Orozco

A thesis submitted to the
Department of Electrical and Computer Engineering
in conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
October 2009

# Abstract

Fountain codes have demonstrated great results for the binary erasure channel and have already been incorporated into several international standards to recover lost packets at the application layer. These include multimedia broadcast/multicast sessions and digital video broadcasting on global internet-protocol. The rateless property of Fountain codes holds great promise for noisy channels. These are more sophisticated mathematical models representing errors on communications links rather than only erasures. The practical implementation of Fountain codes for these channels, however, is hampered by high decoding cost and delay.

In this work we study trapping sets in Fountain codes over noisy channels and their effect on the decoding process. While trapping sets have received much attention for low-density parity-check (LDPC) codes, to our knowledge they have never been fully explored for Fountain codes. Our study takes into account the different code structure and the dynamic nature of Fountain codes. We show that 'error-free' trapping sets exist for Fountain codes. When the decoder is caught in an error-free trapping set it actually has the correct message estimate, but is unable to detect this is the case. Thus, the decoding process continues, increasing the decoding cost and delay for naught. The decoding process for rateless codes consists of one or more decoding

attempts. We show that trapping sets may reappear as part of other trapping sets on subsequent decoding attempts or be defeated by the reception of more symbols. Based on our observations we propose early termination methods that use trapping set detection to obtain improvements in realized rate, latency, and decoding cost for Fountain codes.

# Acknowledgments

I would like to begin by expressing my gratitude towards my supervisor, Prof. Shahram Yousefi, for the opportunity to pursue research under his tutelage. His enthusiasm, discussions, and guidance were of great help in the realization of this thesis. I thank Prof. Juan Carlos Cordova, Prof. Byron Arrivillaga, Prof. Enrique Ruiz and especially my parents, whose examples and clear minds provided me with a love of learning and the impetus to continue my studies. I am also grateful to my alma mater, La Universidad de San Carlos de Guatemala, for embracing me in its culture of ideas and people. I am forever indebted to my friends, both here in Canada and in Guatemala, for the happiness and strength they have given me through their company. I feel extremely fortunate for the care, advice, and love, my godmother, Barb, and my grandmother, Nina, have given me over the years. I can not express how blessed I am to have the endless love and support of my wonderful brother, Tito, and my amazing parents: Roberto Orozco and Vivian Molina. Without them this work would not have been possible. This thesis is dedicated to San Judas Tadeo, my parents, and my brother.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **BEC** | Binary Erasure Channel |
| **BIAWGN** | Binary Input Additive White Gaussian Noise |
| **BP** | Belief Propagation |
| **CSR** | Check Satisfaction Ratio |
| **ID** | Incremental Decoding |
| **i.i.d.** | Independent and Identically Distributed |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IP** | Internet Protocol |
| **LDGM** | Low-Density Generator Matrix |
| **LDPC** | Low-Density Parity-Check |
| **LLR** | Log-Likelihood Ratio |
| **LT** | Luby Transform |
| **MAP** | Maximum A Posteriori |

| | |
|---|---|
| **MRD** | Message Reset Decoding |
| **pdf** | Probability Density Function |
| **TS** | Trapping Set |

# Chapter 1

# Introduction

To communicate is essential. Essential to our self-expression, our collaboration with others, the process of learning, and the formation of knowledge. Due to this, as our world evolves, so also are introduced new ways to communicate. Some of these new methods of communication may have once seemed strange and unconventional, to say the least, but are now quite commonplace. E-mail is one such instance. Transmitting messages consisting of text or multimedia across vast distances in extremely short periods of time is only to be expected.

E-mail, and many other of the new forms of communication (e.g., digital telephony, IP-based video conferencing, text messaging, etc.) come under the realm of digital communications. As with all types of communication, new or old, digital communication involves the transfer of information through a medium, and is thus subject to corruption. A principal objective of communication systems is hence to transmit information as reliably as possible. In digital communications, one way we may protect a message is by encoding it with a channel code. Channel codes add

redundant data to a message, permitting the receiver to detect and correct particular errors. When using channel codes, another of our chief objectives becomes to transmit the information as efficiently as possible, i.e., to not use more redundancy than necessary. Additionally, we also aim for the delay and complexity associated with the encoding and decoding processes to allow practical implementation.

The information rate of a binary channel code is defined as the ratio $R = k/n$, where $k$ is the number of bits in an information message, and $n$ is the number of bits in the codeword transmitted over the channel. Thus, the number of redundant or parity bits added to each message is $(n - k)$. If we have knowledge of the channel conditions at the transmitter, we can select the rate accordingly to obtain both efficient and reliable communication. However, there are many situations where channel conditions vary frequently and are unknown at the transmitter. Regardless, the code rate is often set at a fixed value, resulting in a tradeoff between reliability and efficiency. For example, it is inefficient to set the rate according to a worse case scenario. This implies that most of the time we are adding more redundancy than needed to the transmitted message, wasting valuable channel capacity. On the other hand, the selection of a higher rate implies that communication will sometimes be unreliable.

As opposed to fixed-rate codes, rateless codes adapt their rate to the channel conditions without prior knowledge of the channel statistics. Hence, rateless codes can achieve reliability and efficiency simultaneously, even in situations of channel uncertainty. Nonetheless, most rateless coding schemes have high decoding complexity and may require a large amount of feedback from receiver to transmitter [1–3]. In 2002,

Luby published his remarkable creation [4], Luby Transform (LT) codes, a new class of rateless codes. LT codes were originally designed for the binary erasure channel (BEC), they allow for efficient, reliable, and practical communication for unknown erasure conditions with hardly any feedback. Shortly after, Raptor codes, another great class of rateless codes for the erasure channel were introduced. Raptor codes build on LT codes and were invented by Shokrollahi [5]. Both LT and Raptor codes are members of the class of codes called Fountain codes.

The transmitter of a Fountain code produces a potentially infinite stream of encoded bits to be sent over the channel. For this reason, the transmitter is likened to a fountain continually spouting drops of water. Following the analogy, the receiver is seen as a bucket that collects drops of water until full. It is immaterial to the bucket (receiver) which drops (bits) are collected as long as the amount received is sufficient to fill the bucket (sufficient to infer the original $k$ bits). Once this happens, the receiver requests transmission be stopped. Thus, instead of the rate being set at the transmitter, it is the receiver which dynamically decides the rate depending on the actual instantaneous channel conditions from transmitter to receiver.

Fountain codes can be applied to erasure channels with great results. Networks that use packet-switching such as the Internet, are good real world examples of channels that can be modeled as a BEC. However, there are many other scenarios of channel uncertainty, such as those found in wireless communications, which are not best modeled as erasure channels and which could greatly benefit from the use of rateless codes. Given the excellent properties and results of Fountain codes on the BEC,

there is particular interest in using this class of rateless codes on other channels [6–9]. Unfortunately, obtaining the same performance in terms of reliability, efficiency, and low decoding cost is not straightforward.

For instance, Fountain codes on the BEC have the property of being *universal*, i.e., their realized rate can come very near the channel capacity for any erasure channel. As we might imagine, this property is extremely beneficial for unknown channels. However, in [6] they prove that universal Raptor codes or universal LT codes do not exist for noisy channels (classes of channels beside the erasure channel). Given this, on noisy channels, the design goal for existing classes of Fountain codes is robustness in terms of realized rate for a range of channel conditions which we believe likely to occur.

High decoding cost is another important difference that results when applying Fountain codes to noisy channels. The decoding process used for Fountain codes can be greatly simplified for the binary erasure channel but remains complex for noisy channels. Thus, reducing decoder complexity for Fountain codes is one of the biggest obstacles to overcome before these codes can be used in practical applications over noisy channels.

With Fountain codes a decoding process may consist of several decoding attempts. If the receiver fails to decode on one decoding attempt, it awaits the reception of more symbols and attempts to decode again. On each decoding attempt, iterations of the

belief propagation (BP) algorithm are applied until either decoding is deemed successful or the maximum number of iterations is reached. Early termination of a decoding attempt will clearly result in reduced decoding complexity with negligible performance loss if the decoder is in a state where subsequent iterations will not significantly improve the message estimate.

Early termination of decoding attempts has already been used as a means for complexity reduction in Fountain codes over noisy channels [10, 11]. One of the stopping criteria used for early termination in aforementioned papers is based on the difference between iterations of the check sum satisfaction ratio. Check sums are constraints that the decoder traditionally uses to detect successful decoding. The authors state that the lack of change in the check sum satisfaction ratio implies convergence, and thus early termination may be applied when this occurs. However, they do not delve deeper into this convergent behaviour and relate it to a specific state of the decoder.

In this thesis, we will show that the decoder state that allows for early termination with negligible performance loss is due to the existence of the phenomenon called *trapping sets*. While trapping sets have received much attention for low-density parity-check (LDPC) codes [12–16], to our knowledge they have never been fully explored for Fountain codes over noisy channels. Thus, we will give a definition of trapping sets that is tailored to Fountain codes. This definition is necessary due to the different code structure and the dynamic nature of Fountain codes. We will also show that a new type of trapping sets exists for Fountain codes that does not exist for LDPC codes. We call this new type of trapping sets, 'error-free' trapping sets. The

error-free trapping set is noteworthy and unique. When the decoder is caught in an error-free trapping set it actually has the correct message estimate, but is unable to detect this is the case. Thus, the decoding process continues, increasing the decoding cost and delay for naught.

Through experimental simulation we will obtain results on the amount of decoding failure that is caused by trapping sets, so as to demonstrate their importance on Fountain codes. Moreover, in keeping with the rateless paradigm, we will observe the behaviour of trapping sets not only across the iterations of a single decoding attempt, but across the entire decoding process. We will show that trapping sets may reappear as part of other trapping sets on subsequent decoding attempts or that they may be defeated by the reception of more symbols. Additionally, we will examine the distinct behaviour that error-free trapping sets incur compared to other trapping sets, and the consequences of this.

If the receiver is able to detect when the decoder is caught in a trapping set, then complexity reduction through early termination is possible. While in experiments we can assume knowledge of the transmitted message to enable us to correctly identify the onset of a trapping set, in practice this is not possible. Thus, we also provide results on the accuracy of detecting trapping sets with measurements that are available at the receiver. These measurements are all based on the number of satisfied check sums.

Through our observations on distinct measurements based on check sum satisfaction and their relationship to trapping sets we will propose a method for the early

termination of a decoding attempt. Furthermore, we will propose a new method for the early termination of the entire decoding process. To our knowledge, early termination methods for the entire decoding process had not previously been explored. Thus, we show that through early termination on rateless codes, not only can we reduce decoding complexity, but we may also improve the average realized rate.

An important step towards better designing a good and practical system that implements Fountain codes on noisy channels is understanding the properties of Fountain codes on these channels. To this end, this thesis brings to light, and analyzes the problem of trapping sets in Fountain codes over noisy channels.

## 1.1 Thesis Outline

In Chapter 2, we present fundamental concepts that will be used throughout the thesis. In particular, we elaborate on the decoding process used in Fountain codes. In Chapter 3, we discuss trapping sets in Fountain codes over noisy channels. We begin by giving a definition of trapping sets that is suitable to Fountain codes and will allow their proper study. Within this framework we show the existence of a new type of trapping set which is error-free. We then present our experimental results on the appearance of trapping sets in Fountain codes while decoding. These include our observations on the behaviour of trapping sets across the entire decoding process. Thus showing how the rateless property of Fountain codes affects trapping sets. Additionally, we show how we can use measurements available at the decoder to detect trapping sets. Having already examined trapping sets, we discuss how we can exploit

this knowledge for early termination of decoding attempts or of the complete decoding process. Early termination methods address the need for complexity reduction of Fountain codes on noisy channels. Moreover, we show that they can also be used to obtain improvements in average realized rate. Based on our observations we propose two early termination methods. Finally, we present some conclusions and ideas for future work in Chapter 4.

## 1.2 Thesis Contributions

The main contributions of this thesis are:

- The discovery of error-free trapping sets in Fountain codes, and for the general class of low-density generator matrix (LDGM) codes.
- The study of the effect of ratelessness on trapping sets in Fountain codes. Specifically, we show that a trapping set encountered during one decoding attempt may be part of the trapping set encountered during the next decoding attempt. Furthermore, we show that the reception of more bits on the next decoding attempt may allow the decoder to avoid a trapping set.
- Methods to improve the complexity and realized rate of Fountain codes. These methods use early termination when a trapping set has been detected at the decoder.

# Chapter 2

# Fundamentals

## 2.1 Digital Communication System

In Figure 2.1, we present a model of a digital communication system which has been simplified to show that our main concern is on channel coding. Of the transmitter of the communication system, we are only interested in the function block of the encoder, similarly, with the receiver we only present its decoder function block. For the purposes of this work we assume a binary memoryless symmetric source. That is, we assume the source generates a sequence of independent and identically distributed (i.i.d.) random variables taking values in a binary alphabet with equal probability. We will use the binary alphabet {+1, -1} instead of {0, 1}, with 0 mapping to +1 and 1 mapping to -1. Additionally, we assume a binary code, thus, each codeword output from the encoder also consists of members of the Galois Field GF(2).

The information sequence from the digital source is divided into message blocks of $k$ information bits each. The message block, or simply message, is then $\mathbf{m} =$

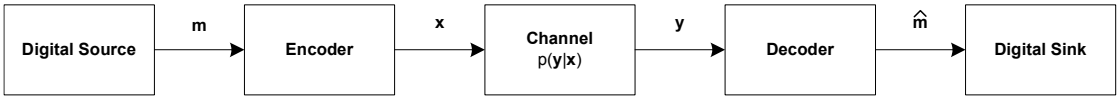



Figure 2.1: Simplified block diagram of a digital communication system

$(m_1, m_2, \ldots, m_k)$. The message $\mathbf{m}$ is transformed by the encoder into a codeword $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of $n$ bits. The codeword $\mathbf{x}$ is then sent across the channel and corrupted by noise, resulting in a received sequence $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. We must keep in mind that depending on the noise introduced by the channel the received sequence $\mathbf{y}$ may no longer consist of discrete values from the binary alphabet. The decoder transforms the received sequence $\mathbf{y}$ into an estimate of the message, $\hat{\mathbf{m}} = (\hat{m}_1, \hat{m}_2, \ldots, \hat{m}_k)$. Ideally, $\hat{\mathbf{m}} = \mathbf{m}$, but channel noise may cause some decoding errors.

We will use the notation $\mathcal{I}_M$ for a set of discrete numbers $\{1, 2, \ldots, M\}$. Each $m_i$, where $i \in \mathcal{I}_k$, represents a single information bit. Each $x_j$, where $j \in \mathcal{I}_n$ represents a single encoded bit. Each $y_j$, where $j \in \mathcal{I}_n$, represents a single received symbol.

## 2.2 Channels

A channel can be characterized by its input and output alphabets and its conditional probability density function (pdf), $p(\mathbf{y}|\mathbf{x})$. The expression $p(\mathbf{y}|\mathbf{x})$ gives us the probability that the channel output sequence is $\mathbf{y}$ given that the channel input sequence was $\mathbf{x}$ . We will assume that the channel is memoryless, i.e., the output at any time instant depends only on the input at that time instant. We can then describe the channel by the conditional pdf of a single time instant, $p(y|x)$, the probability that symbol $y$ is observed given that bit $x$ was sent. Due to the memoryless property of

the channel, $p(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{n} p(y_j|x_j)$.

While there are many different channels, in this work we are concerned mainly with the binary input additive white Gaussian noise (BIAWGN) channel. In what follows we describe this channel as well as the binary erasure channel for which Fountain codes were originally conceived.

### 2.2.1 Binary Erasure Channel

In the binary erasure channel (BEC), the receiver is either completely certain or completely uncertain of what the transmitted bit was. When the latter occurs, we say the bit is lost or erased. The probability of a bit being erased is $p$, and that of it being transmitted perfectly through the channel is $1-p$. Figure 2.2 shows a mapping from channel input X to channel output Y. As shown, this channel has a binary input alphabet and a ternary output alphabet.

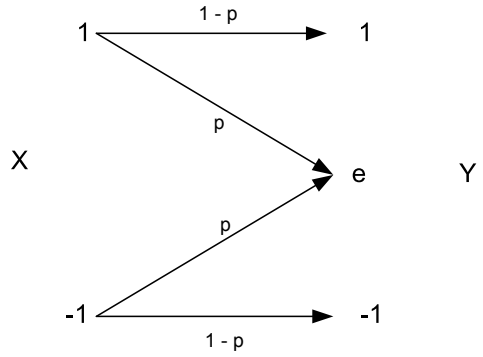



Figure 2.2: The binary erasure channel

### 2.2.2 Binary Input Additive White Gaussian Noise Channel

The BIAWGN channel has a binary input alphabet, +1 and -1, and a continuous output alphabet, the set of real numbers. The channel output $Y$ is given by $X + N$, where $X$ is the channel input and $N$ is added noise. $N$ is a memoryless Gaussian random variable with zero mean and variance $\sigma^2$. Hence, the conditional pdf $p(y|x)$ is a Gaussian pdf with mean $x$ and variance $\sigma^2$, as seen in equation (2.1):

$$p(y|x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right). \tag{2.1}$$

The ratio $E_S/\sigma^2$ of signal energy per transmitted codeword bit, $E_S$, to noise energy, $\sigma^2$, is often used to compare coding schemes for Gaussian channels. For our setting we have $E_S = 1$, since $X$ belongs to $\{+1, -1\}$. We may also use the ratio $E_S/N_0$, where $N_0 = 2\sigma^2$ is the double-sided power spectral density. Another alternative is $E_b/N_0$, where $E_b$ is the energy per transmitted information bit. The relationship between $E_b$ and $E_S$ is $E_b = E_S/R$, where $R$ is the code rate.

## 2.3 Decoder

Decoders are algorithms that aim to solve a problem in statistical inference. Given the received sequence $\mathbf{y}$, the decoder must infer the transmitted message $\mathbf{m}$.

Based on Bayesian detection theory, there are two main strategies for decoders to produce the message estimate $\hat{\mathbf{m}}$ given $\mathbf{y}$ depending on the probability of error they wish to minimize.

### 2.3.1 Block MAP Decoding

In block maximum a posteriori probability (MAP) decoding the decoder aims to minimize the probability of block error. To this purpose, it must minimize $p(\hat{\mathbf{m}} \neq \mathbf{m}|\mathbf{y})$.

Besides producing an estimate $\hat{\mathbf{m}}$ of the message $\mathbf{m}$, the decoder can also produce an estimate $\hat{\mathbf{x}}$ of $\mathbf{x}$. Since there is a one-to-one correspondence between the message block $\mathbf{m}$ and the codeword $\mathbf{x}$, then $\hat{\mathbf{m}} = \mathbf{m}$ if and only if $\hat{\mathbf{x}} = \mathbf{x}$. Hence, we generally state that the block MAP decoder aims to minimize $p(\hat{\mathbf{x}} \neq \mathbf{x}|\mathbf{y})$. This is equivalent to maximizing the a posteriori probability $p(\hat{\mathbf{x}} = \mathbf{x}|\mathbf{y})$. Thus, in block MAP decoding we choose $\hat{\mathbf{x}}$ to be the codeword $\mathbf{x}$ that maximizes $p(\mathbf{x}|\mathbf{y})$.

The block MAP decoding rule $\hat{\mathbf{x}}^{\text{MAP}}(\mathbf{y})$ for a code $\mathcal{C}$ is:

$$\begin{aligned}
\hat{\mathbf{x}}^{\text{MAP}}(\mathbf{y}) &= \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{y}) && (2.2)\\
&= \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{x})\frac{p(\mathbf{x})}{p(\mathbf{y})} \qquad \text{(Bayes' theorem)} \\
&= \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{x})p(\mathbf{x}). && (2.3)
\end{aligned}$$

The received sequence $\mathbf{y}$ is a constant given as input to the decoder. We have used $\mathbf{x} \in \mathcal{C}$ to denote that $\mathbf{x}$ belongs to the set of valid codewords of $\mathcal{C}$. If all codewords of $\mathcal{C}$ are equally likely, that is, the a priori probability $p(\mathbf{x})$ is uniform, then we say the block MAP decoder is a *maximum likelihood* (ML) decoder. Thus, Equation (2.3)

becomes:

$$\hat{\mathbf{x}}^{\mathrm{MAP}}(\mathbf{y}) = \operatorname*{argmax}_{\mathbf{x}\in\mathcal{C}} p(\mathbf{y}|\mathbf{x}) = \hat{\mathbf{x}}^{\mathrm{ML}}(\mathbf{y}). \tag{2.4}$$

### 2.3.2 Bitwise MAP Decoding

The objective of the bitwise MAP decoder is to minimize the probability of bit error. For this, it must minimize the probability $p(\hat{m}_i \neq m_i|\mathbf{y})$ for $i \in \mathcal{I}_k$. This is equivalent to maximizing the a posteriori probability $p(\hat{m}_i = m_i|\mathbf{y})$ that each information bit $m_i$ is correctly decoded. Thus, in bitwise MAP decoding we choose $\hat{m}_i$ to be the value in GF(2) that maximizes $p(m_i|\mathbf{y})$.

The bitwise MAP decoding function $\hat{m}_i^{\mathrm{MAP}}(\mathbf{y})$ can be expressed as follows[1]:

$$\begin{aligned}
\hat{m}_i^{\mathrm{MAP}}(\mathbf{y}) &= \operatorname*{argmax}_{m_i\in\{+1,-1\}} p(m_i|\mathbf{y}) && (2.5)\\
&= \operatorname*{argmax}_{m_i\in\{+1,-1\}} \sum_{\sim m_i} p(\mathbf{m}|\mathbf{y}) \qquad \text{(law of total probability)}\\
&= \operatorname*{argmax}_{m_i\in\{+1,-1\}} \frac{1}{p(\mathbf{y})} \sum_{\sim m_i} p(\mathbf{y}|\mathbf{m})p(\mathbf{m}) \qquad \text{(Bayes' theorem)}\\
&= \operatorname*{argmax}_{m_i\in\{+1,-1\}} \sum_{\sim m_i} p(\mathbf{y}|\mathbf{m})p(\mathbf{m}). && (2.6)
\end{aligned}$$

The notation $\sum_{\sim m_i}$ denotes a summation (marginalization) over all the components of $\mathbf{m}$ except $m_i$. That is, $\sum_{\sim m_i} = \sum_{m_1}\cdots\sum_{m_{i-1}}\sum_{m_{i+1}}\cdots\sum_{m_k}$.

[1]Some decoders (e.g., LDPC codes) perform bitwise MAP decoding or approximate bitwise MAP decoding for the codeword bits rather than the message bits, i.e., their decoding rule is $\hat{x}_i^{\mathrm{MAP}}(\mathbf{y}) = \operatorname{argmax}_{x_i\in\{+1,-1\}} p(x_i|\mathbf{y})$.

### 2.3.3 Log-Likelihood Ratios

We denote the log-likelihood ratio (LLR) of a binary random variable $u$ as $L(u)$, where

$$L(u) = \log \frac{p(u=+1)}{p(u=-1)}.$$

Log-likelihood ratios provide a way of comparing the two probabilities of a binary random variable. If $L(u) = 0$, then it is equiprobable for $u$ to be equal to +1 or -1. If $L(u) > 0$, $u = +1$ is more likely, else, if $L(u) < 0$, $u = -1$ is more likely. The sign of the LLR is the hard decision on the value of the binary random variable, and the magnitude of the LLR is the reliability of this decision.

If the binary random variable $u$ is conditioned on another variable or vector $\mathbf{z}$, then we have a conditional log-likelihood ratio $L(u|\mathbf{z})$. A conditional LLR is useful in expressing the channel information of a transmitted bit $x$ given that a symbol $y$ was received. The channel LLR is defined as

$$\begin{aligned} L(x|y) &= \log \frac{p(x=+1|y)}{p(x=-1|y)} \\ &= \log \left( \frac{p(y|x=+1)}{p(y|x=-1)} \cdot \frac{p(x=+1)}{p(x=-1)} \right) \\ &= \log \frac{p(y|x=+1)}{p(y|x=-1)}) + \log \frac{p(x=+1)}{p(x=-1)} \\ &= \log \frac{p(y|x=+1)}{p(y|x=-1)} + L(x) \\ &= \log \frac{p(y|x=+1)}{p(y|x=-1)}. \end{aligned} \tag{2.7}$$

In (2.7) we have used the assumption that the a priori probabilities of $x$ are equal, and therefore $L(x) = 0$.

For the BEC channel,

$$L(x|y) = \log p(y|x = +1) - \log p(y|x = -1) = \infty \cdot y.$$

For the BIAWGN channel,

$$L(x|y) = \log \frac{\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right)}{\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right)} = \frac{2}{\sigma^2} \cdot y.$$

It is often convenient to express the bitwise MAP decoding task in terms of LLRs. Thus, we should calculate the conditional LLR $L(m_i|\mathbf{y})$ and base our hard decision $\hat{m}_i$ on the sign of the LLR for all $i \in \mathcal{I}_k$.

## 2.4 Fountain Codes

For a given message block $\mathbf{m} = (m_1, m_2, \ldots, m_k)$, a Fountain code produces a potentially infinite stream $\mathbf{x} = (x_1, x_2, \ldots)$ of encoded bits to be sent over the channel. Each encoded bit is generated independently by sampling from a fixed distribution on subsets of the message block and then performing addition modulo-2 on the information bits in the chosen subset.

The encoder continues to produce more encoded bits until the decoder acknowledges it has recovered the message block. Thus, the number of encoded bits, $n$, in the final codeword is not known beforehand[2]. We say then that the Fountain code

[2]For Fountain codes, $n$ is generally defined as the number of *received* encoded bits. This is especially important to keep in mind for the erasure channel where bits may be lost.

used is generated on-the-fly or in an *online* fashion. This allows the message blocks encoded by a Fountain encoder to have different code rates.

A Fountain code used may be represented by a dynamic generator matrix $\mathbf{G}$. The generator matrix has $k$ rows, one for each information bit, and $n$ columns, one for each encoded bit. However, as mentioned previously, the value of $n$ is not fixed a-priori. Each time an encoded bit is generated, a new column is added to the generator matrix. A message bit $m_i$ belongs to the subset of the message block that was added modulo-2 to produce the encoded bit $x_j$ if and only if the element at the intersection of row $i$ and column $j$ of the generator matrix is non-zero. The codeword $\mathbf{x}$ at a given time can be given as $\mathbf{x} = \mathbf{mG'}$. Where $\mathbf{G'}$ is the generator matrix in GF(2) using the set $\{+1, -1\}$ instead of the set $\{0, 1\}$. In Figure 2.3, a snapshot of a Fountain code's generator matrix $\mathbf{G}$ is shown.

$$
\begin{array}{c} \\ m_1 \\ m_2 \\ \vdots \\ m_k \end{array}
\begin{array}{c}
\begin{array}{ccccc} x_1 & x_2 & x_3 & \cdots & x_n \end{array} \\
\left[\begin{array}{ccccc}
1 & 0 & 1 & \cdots & 0 \\
0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 1 & 0 & \cdots & 1
\end{array}\right]
\end{array}
$$

Figure 2.3: Generator matrix $\mathbf{G}$ of a Fountain code.

The *weight* of a column in a matrix is defined as the number of its non-zero elements. A Fountain code's generator matrix $\mathbf{G}$ is *sparse*, if the average weight of a column is small compared to $k$. Fountain codes are generally designed to produce a sparse generator matrix, since this results in low encoding and decoding cost. We

consider a sparse Fountain code to be a rateless low-density generator matrix (LDGM) code[3].

### 2.4.1 Luby Transform (LT) Codes

For Fountain codes the encoding operation is identical for each information block. Encoded bits are generated sequentially from a subset of the information bits and sent over the channel. For the class of Fountain codes called LT codes, the distribution on subsets of the message block is induced by a *degree distribution* [18].

LT codes are Fountain codes specified by a degree distribution which generate encoded bits using the following three steps: First, a value $d$ is drawn from a distribution $\Omega_1, \Omega_2, \ldots, \Omega_k$ over $\mathcal{I}_k$, where $\Omega_i$ is the probability that $i$ is chosen. Afterwards, $d$ distinct information bits are chosen uniformly at random from the message block. In the final step, modulo-2 addition is performed over the selected information bits. The result of this operation is the value of the encoded bit to be sent over the channel.

Following a graph-theoretic terminology, the $d$ information bits which are added modulo-2 together to form an encoded bit are called its neighbours. The value $d$ associated with an encoded bit is called its degree, and the distribution from which it is sampled, the *degree distribution*. The degree distribution may also be represented by its generator polynomial $\Omega(v) = \sum_{i=1}^{k} \Omega_i v^i$. Several system parameters depend on the choice of the degree distribution. These include encoding cost, decoding cost,

[3]Fixed-rate LDGM codes have been studied for source coding, channel coding, and statistical physics [17].

average realized rate, error performance, and the ability to successfully start and continue the decoding process. Hence, degree distribution design is critical and has been studied in such works as [4, 6, 18–20].

In the generator matrix $\mathbf{G}$ of a Fountain code, non-zero elements in a column are the neighbours of the corresponding encoded bit. Moreover, the weight of a column is the same as its degree. To reduce encoding and decoding cost, degree distributions for LT codes are designed to produce encoded bits with as low an average degree as possible.

### 2.4.2 Raptor Codes

Raptor codes concatenate an LT code with a precode. The precode is a fixed-rate code that transforms a message $\mathbf{m}$ of $k$ bits into an intermediate codeword $\tilde{\mathbf{m}}$ of $k' > k$ bits. These $k'$ bits are input into the LT encoder to produce a codeword $\mathbf{x}$ of undetermined length $n$. Hence, it is the LT component of a Raptor code which allows the overall code to be a rateless code. For Raptor codes, we call the $k'$ bits of the intermediate codeword, input bits, and the $n$ bits of the final codeword, output bits.

As discussed, for Raptor codes encoding is realized in two phases: one phase corresponds to the precode and the other to the LT code. For this reason, in this work, we will think of Raptor codes as being a *precoded Fountain code* rather than just a Fountain code.

The precode of a Raptor code is generally chosen to be a high rate LDPC code.

The generator matrix of an LDPC code is not necessarily sparse, but may still possess a linear time encoding algorithm [21]. Additionally, LDPC codes possess an efficient decoding algorithm and have good performance over several different channels [22].

## 2.5 Fountain Decoder

The decoder plays an important role in a code's performance and cost. Moreover, the problem of trapping sets which we study in this work is encountered during the decoding process. The decoding process is the set of tasks through which the decoder transforms a received sequence $\mathbf{y}$ of length $n$ into a message estimate $\hat{\mathbf{m}}$.

For Fountain codes a constant stream of encoded bits is being output from the encoder and sent over the channel. Hence, the length of the received sequence grows with time. The decoding process for Fountain codes consists of a series of decoding attempts $(1, 2, \ldots)$, which correspond to received sequences $(\mathbf{y_1}, \mathbf{y_2}, \ldots)$ of lengths $(n_1, n_2, \ldots)$, where $n_1 < n_2 < \ldots$ . The number of decoding attempts that is necessary to complete the decoding process is unknown a priori. We denote the received sequence at the final decoding attempt (i.e., at the end of the decoding process) as $\mathbf{y}$. Additionally, we denote the length of the final received sequence as $n$.

A decoding attempt $a$ aims to obtain a message estimate $\hat{\mathbf{m}}$ that satisfies a criterion for successful decoding given a received sequence $\mathbf{y}_a$. If decoding at attempt $a$ fails to meet the criterion for success, then the decoder awaits the reception of the $(n_{a+1} - n_a)$ symbols necessary for a received sequence of length $n_{a+1}$ and then performs the next decoding attempt. On the other hand, if decoding at attempt $a$

meets the criterion for success, then the final received sequence $\mathbf{y}$ is equal to $\mathbf{y}_a$ and the length $n$ of the final received sequence is equal to $n_a$. When a successful decoding attempt occurs, the decoding process terminates and the receiver requests that the transmission of more encoded bits be stopped.

The first decoding attempt begins after $n_1$ symbols have been received, where $n_1$ is usually chosen such that $k/n_1$ is equal to or slightly lower than the realized capacity of the channel. For the purposes of this work, we assume the number of symbols the receiver waits to receive between decoding attempts is a constant $T$. Hence, the number of symbols received at decoding attempt $a$ is $n_a = n_1 + (a-1)T$.

Each decoding attempt uses the iterative belief propagation algorithm, which approximates bitwise MAP decoding, to try and obtain a successful message estimate $\hat{\mathbf{m}}$. The decoder has a finite number $l_{max}$ of iterations in which to succeed for a given decoding attempt. If the maximum number of iterations is realized without success, the decoding attempt is marked as a failure and the decoder awaits the reception of more symbols to perform the next decoding attempt.

Ideally, we would be able to perform an infinite number of decoding attempts and have a received sequence of infinite length. In practice, however, the encoder is limited to generating a finite number $n_{max}$ of encoded bits to be transmitted over the channel. That is, the minimum allowable rate is set to $k/n_{max}$. If we do not limit the allowable number of transmitted encoded bits per message block, a decoding process may terminate at a value of $n$ that results in a tiny realized rate with impractically

large associated delay or not terminate at all.

If the decoding process reaches the maximum allowable number of attempts and iterations without success, then the decoding process terminates and is declared a failure. The message estimate $\hat{\mathbf{m}}$ available at that time is output from the decoder.

In the following section we give a general description of the belief propagation algorithm, since this is used on all decoding attempts. Afterwards, we discuss in more detail the specifics of the decoding algorithm for Fountain codes.

### 2.5.1 Belief Propagation Decoding

For bitwise MAP decoding we must obtain marginals of the form

$$p(m_i|\mathbf{y}) = \mathbb{C} \cdot \sum_{m_1} \cdots \sum_{m_{i-1}} \sum_{m_{i+1}} \cdots \sum_{m_k} p(\mathbf{y}|m_1, \ldots, m_k),$$

where $\mathbb{C}$ is some constant. For most codes performing bitwise MAP decoding in this form is intractable. However, we can look at $\mathbb{C} \cdot p(\mathbf{y}|m_1, \ldots, m_k)$ as a function $g(m_1, \ldots, m_k)$ which can be factored as $\prod_a f_a$, where each $f_a$ is a local function depending only on a subset of $\{m_1, \ldots, m_k\}$. Thus, not all factors need be summed over all the combinations that the values of the arguments of $g$ can take (excluding the argument $m_i$).

Factorization is a *divide and conquer* approach to bitwise MAP decoding. As an

example, assume we wish to obtain $\hat{m}_1$ using bitwise MAP decoding and that

$$\begin{aligned} p(m_1|\mathbf{y}) &= \sum_{m_2}\sum_{m_3}\sum_{m_4} g_1(m_1, m_2, m_3, m_4) \\ &= \sum_{m_2}\sum_{m_3}\sum_{m_4} f_A(m_1) f_B(m_1, m_2) f_C(m_2, m_4) f_D(m_1, m_3, m_4). \end{aligned}$$

We may rewrite this as

$$p(m_1|\mathbf{y}) = f_A(m_1) \sum_{m_2} f_B(m_1, m_2) \sum_{m_4} f_C(m_2, m_4) \sum_{m_3} f_D(m_1, m_3, m_4).$$

In this factorized form we may perform operations locally. For example, we may first realize the operation $\sum_{m_3} f_D(m_1, m_3, m_4)$ and then use this result for further calculations.

A graph, appropriately named *factor graph*, can be used to elegantly model the factorization structure of a function of many variables.

**Definition 2.1 :** A factor graph is a graphical representation of a factorization

$$g(v_1, v_2, \ldots, v_M) = \prod_a f_a,$$

where each $f_a$ is a local function depending only on a subset of $\{v_1, \ldots, v_M\}$. A factor graph is a bipartite graph whose nodes (vertices) can be divided into two disjoint sets called function nodes and variable nodes. Every edge in the factor graph connects a function node to a variable node. The factor graph has a variable node for each variable $v_i$ and a function node for each local function $f_a$. An edge joins the variable node $v_i$ and the function node $f_a$ if and only if

the variable $v_i$ is among the arguments of $f_a$.

Bitwise MAP decoding may be performed on a factor graph using a *message passing algorithm*. In the message passing algorithm, operations are performed locally at nodes of the factor graph. Afterwards, the results (messages) are sent through the edges of the factor graph to neighbouring nodes for further calculations. For bitwise MAP decoding the messages sent along the factor graph are based on probabilities. Consequently, when a message passing algorithm[4] is used to perform bitwise MAP decoding, it is referred to as the belief propagation (BP) algorithm. The computation of a marginal $p(m_i|\mathbf{y})$ is only exact when the underlying factor graph is cycle-free[5]. This is generally not the case. Thus, we say that BP decoding is an *approximate* bitwise MAP decoding.

The decoder must obtain marginals $p(m_i|\mathbf{y})$ for all $m_i$, where $i \in \mathcal{I}_k$. The computation of these marginals involves many common terms. When using the belief propagation algorithm the approximation of these marginals may all be computed simultaneously on the same factor graph. Hence, the BP algorithm is an efficient decoding algorithm based on the principle of information/calculation reuse.

### Parity Check Equations

The factor graph for the BP decoding algorithm not only models the factorization of the a posteriori joint probability density function of the variables $\mathbf{m} = (m_1, \ldots, m_k)$ given the channel output $\mathbf{y}$, but also the valid behaviour of variable

[4]For a more in-depth discussion on the message passing algorithm confer [17, 23, 24].

[5]We define a cycle in a factor graph as a closed path, i.e., a finite set of connected edges which starts and ends at the same node, where no node (except the starting and ending node) appears more than once.

configurations. For LDGM codes this valid behaviour can be expressed by the indicator function $\delta[\mathbf{mG'} = \mathbf{x}]$ [6]. The indicator function evaluates to 1 on sequences $\mathbf{m} = (m_1, m_2, \ldots, m_k)$ for which $\mathbf{mG'} = \mathbf{x}$, and to 0 otherwise.

The indicator function may be viewed as a codeword membership function, where $\mathbf{x}$ is a valid codeword if and only if there exists a message $\mathbf{m}$ such that $\mathbf{mG'} = \mathbf{x}$. Additionally, we may view the indicator function as a scaled version of the uniform distribution over all message blocks, $p(\mathbf{m}) = (1/2^k) \cdot \delta[\mathbf{mG'} = \mathbf{x}]$, which may be substituted in Equation (2.6).

To represent the indicator function on the factor graph, we factorize it into a product of smaller local indicator functions. Each local indicator function includes only a subset of $\{x_1, x_2, \ldots, x_n, m_1, m_2, \ldots, m_k\}$ as arguments. The product of the local indicator function evaluates to 1 only if all functions evaluate to 1. We recall that the value of each encoded bit $x_j$ is the sum modulo-2 of its neighbours. Hence, we factorize the indicator function as

$$\delta[\mathbf{mG'} = \mathbf{x}] = \prod_{j=1}^{n} \delta[\bigoplus_{m_l \in \mathcal{N}(x_j)} m_l = x_j].$$

We will use the notation $\mathcal{N}(x_j)$ to represent the set of neighbours of $x_j$, $\oplus$ to represent modulo-2 addition, and $\bigoplus$ to denote the summation operator for modulo-2 addition.

Furthermore, since modulo-2 addition is identical to modulo-2 subtraction, we

[6] For LDPC codes the indicator function is based on the code's parity-check matrix $\mathbf{H}$ and defined as $\delta[\mathbf{xH^T} = \mathbf{0}]$.

may write

$$\delta[\mathbf{mG}' = \mathbf{x}] = \prod_{j=1}^{n} \delta[x_j \oplus \bigoplus_{m_l \in \mathcal{N}(x_j)} m_l = +1],$$

where $+1$ is the null element under modulo-2 addition in GF(2) with the elements $\{+1, -1\}$. The equations $x_j \oplus \bigoplus_{m_l \in \mathcal{N}(x_j)} m_l = +1$ may be viewed as parity check equations, which ensure that the members of the set $\{x_j, \mathcal{N}(x_j)\}$ have an even number of $-1$ values.

As mentioned previously, the neighbours of an encoded bit are the nonzero values in the corresponding column of the generator matrix $\mathbf{G}$. Thus, the generator matrix provides us with a convenient representation of the subset of message bits upon which each local indicator function depends. Fountain codes decode using the belief propagation algorithm upon a factor graph based on the code's generator matrix $\mathbf{G}$. We assume the decoder has knowledge of $\mathbf{G}$. Since Fountain codes are generated on-the-fly, different methods are available for this. For example, the decoder may reproduce the generator matrix using the identical pseudo-random realization used at the encoder [4].

### 2.5.2 Fountain Decoding Graph

In Figure 2.4, we depict an example factor graph for a Fountain code at a decoding attempt $a$. The square nodes are function nodes and the circle-shaped nodes are variable nodes. We will refer to the function nodes as check nodes, as they represent the local parity check equations.

The factor graph for a Fountain code contains two types of variable nodes: message nodes, which represent message bits, and encoded nodes, which represent encoded bits. Connecting to each check node from the left hand side are the subset of message nodes that were added modulo-2 together to produce the encoded node connected to the right hand side of the check node.

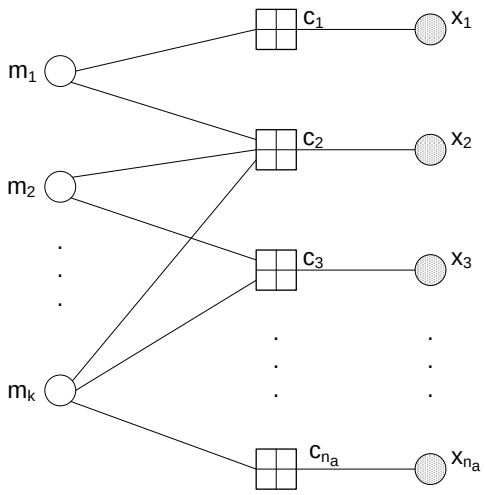



Figure 2.4: Fountain decoding graph

In the figure, the check nodes have been labeled as $c_j$, where $j \in \mathcal{I}_{n_a}$. The neighbourhood of a check node $c_j$, denoted as $\mathcal{N}(c_j)$, is the set of *message* nodes to which it has connections. The degree of a check node is $|\mathcal{N}(c_j)|$. The neighbourhood of a message node $m_i$, denoted as $\mathcal{N}(m_i)$, is the set of check nodes to which it is connected. The degree of a message node is $|\mathcal{N}(m_i)|$. All encoded nodes have only one neighbouring check node, and hence are of degree 1. We note that the definition of degree for the encoded node in the decoding process is not the same as that for the encoded bit during the encoding process.

### 2.5.3 Fountain Decoding Matrix

The decoding graph of a Fountain code is generator matrix based. The message nodes and check nodes of the decoding graph are represented by rows and columns of the generator matrix, respectively. Again, we note that this definition differs from the one used during the encoding process, in which columns of the generator matrix represented encoded bits. An edge connects a message node $m_i$ to a check node $c_j$ if and only if the element at the intersection of row $i$ and column $j$ of the generator matrix $\mathbf{G}$ is nonzero.

However, the generator matrix alone does not completely describe the Fountain decoding graph. We must also show the connections between check nodes and encoded nodes. For this, we will append below the $k$-by-$n_a$ generator matrix an $n_a$-by-$n_a$ identity matrix. The rows of the identity matrix represent encoded nodes and the columns represent check nodes. Thus, each check node has only one edge to an encoded node and vice versa. In this complete matrix representation, variable nodes (both message nodes and encoded nodes) are represented as rows and check nodes as columns. Generally, the factor graph is defined only in terms of the generator matrix, but our representation will be useful later on when defining trapping sets in Fountain codes.

In Figure 2.5 we show a decoding matrix that represents the decoding graph of Figure 2.4 . Above the horizontal line is the generator matrix and below it the identity matrix.

$$
\begin{array}{c}
 \\
m_1 \\
m_2 \\
\vdots \\
m_k \\
 \\
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_{n_a}
\end{array}
\begin{array}{c}
\begin{array}{ccccc} c_1 & c_2 & c_3 & \cdots & c_{n_a} \end{array} \\
\left[\begin{array}{ccccc}
1 & 1 & 0 & \cdots & 0 \\
0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 1 & 1 & \cdots & 1 \\
\hline
1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{array}\right]
\end{array}
$$

Figure 2.5: Fountain decoding matrix.

Since Fountain codes are rateless, the decoding graph and the decoding matrix will grow on each decoding attempt. If $T$ more encoded bits are received for another decoding attempt, then $T$ more columns are added to the generator matrix, and $T$ more columns and rows are added to the identity matrix within the decoding matrix.

### 2.5.4 Fountain Decoding Algorithm

Due to the rateless property of Fountain codes, the decoding process for a single codeword may consist of several decoding attempts. At each decoding attempt, iterative belief propagation decoding is performed.

For each decoding attempt, following a set of simple processing rules, the belief propagation algorithm approximately computes marginal functions of the form $p(m_i = +1|\mathbf{y})$ and $p(m_i = -1|\mathbf{y})$ derived from a global function. Although in the BP

decoding algorithm we may send messages consisting of 2-tuples of probabilities, it is convenient to send LLRs instead. Our objective then becomes to use the BP algorithm to calculate the pseudo[7] a posteriori LLR $L(m_i|\mathbf{y})$ for all $i \in \mathcal{I}_k$. Because the messages are log-likelihoods, the processing rules at nodes of the factor graph take on a different form than they would otherwise. For instance, multiplications are mapped to additions. Furthermore, the processing rules are derived using LLR-algebra, introduced by [25] to compute the LLR of the sum modulo-2 of variables, e.g., $L(u_1 \oplus u_2)$.

The BP decoding algorithm proceeds in iterations. For each iteration, messages in the form of pseudo LLRs are passed from message nodes to check nodes and vice versa. We denote a message node as $m_i$, where $i \in \mathcal{I}_k$. A check node is denoted as $c_j$, where $j \in \mathcal{I}_{n_a}$ and $n_a$ is the number of received symbols at decoding attempt $a$. A message sent from a message node $m_i$ to a check node $c_j$ at the $l$th iteration is represented as $M^{(l)}_{m_i \to c_j}$, likewise, a message sent from a check node $c_j$ to a message node $m_i$ is represented as $M^{(l)}_{c_j \to m_i}$. Each encoded node $x_j$ sends a constant message to its neighbouring check node $c_j$. This message is the channel LLR $L(x_j|y_j)$, which we will denote as $Z_j$ . The notation $\mathcal{N}(w)\backslash\{v\}$ denotes all the neighbors of a node $w$ other than $v$.

On the first decoding attempt and first iteration, all message nodes send messages to all their neighbouring check nodes with a value of 0. This is due to the fact that the message nodes have received no information at this point. Afterwards in the attempt,

[7] We have described the LLR as a pseudo LLR as it is an estimate of the exact LLR due to the existence of cycles in the graph.

messages are calculated as follows:

$$M_{c_j \to m_i}^{(l)} = 2 \cdot \operatorname{arctanh}\left(\tanh\left(\frac{Z_j}{2}\right) \cdot \prod_{m_i' \in \mathcal{N}(c_j)\setminus\{m_i\}} \tanh\left(\frac{M_{m_i' \to c_j}^{(l)}}{2}\right)\right), \tag{2.8}$$

$$M_{m_i \to c_j}^{(l+1)} = \sum_{c_j' \in \mathcal{N}(m_i)\setminus\{c_j\}} M_{c_j' \to m_i}^{(l)}. \tag{2.9}$$

After running the BP algorithm for $l$ iterations, the pseudo a posteriori LLR of each message bit $m_i$ can be calculated as the sum $\sum_{c_j \in \mathcal{N}(m_i)} M_{c_j \to m_i}^{(l)}$, where the sum is over *all* the neighbours of $m_i$ . A message estimate $\hat{\mathbf{m}} = (\hat{m}_1, \hat{m}_2, ..., \hat{m}_k)$ can be produced through a hard decision on the pseudo a posteriori LLRs.

If the message estimate $\hat{\mathbf{m}}$ at a given iteration satisfies a success criterion, then both the decoding attempt and the decoding process terminate. Otherwise, the decoding attempt continues until the maximum number of iterations has been reached. If at this point, the decoding attempt has not succeeded, then another decoding attempt will be realized after the reception of more symbols.

The initialization of the BP algorithm may vary on decoding attempts subsequent to the first attempt. The traditional decoding algorithm used on Fountain codes is *message reset decoding* (MRD). In MRD, messages are always initialized to 0 at the beginning of any decoding attempt, thereafter, Equations (2.8) and (2.9) are used to

update messages. In [26, 27] *incremental decoding* (ID) was introduced for Fountain codes. Whereas MRD always begins from scratch, ID continues decoding using the messages from the previous attempt. This allows for faster convergence of the BP decoding algorithm on attempts $a > 1$.

Additionally, there are different flavours of message update scheduling for the BP algorithm. Here we have described the traditional parallel or *flooding schedule*. In the flooding schedule all messages $M^{(l)}_{m_i \to c_j}$ are updated simultaneously and afterwards all messages $M^{(l)}_{c_j \to m_i}$ are updated simultaneously. However, other scheduling techniques update only a fraction of the variable-to-check messages followed by the updating of a few check-to-variable messages. In [10] different scheduling techniques are applied to Fountain codes.

### Stopping Criteria

For Fountain codes, two different stopping criteria are necessary: a stopping criteria for decoding attempts and a stopping criteria for the complete decoding process [11]. For the decoding attempt, one of the stopping criteria is always whether the maximum number $l_{max}$ of iterations has been reached. For the entire decoding process, one of the stopping criteria is always whether we have reached the maximum number $l_{max}$ of iterations on a decoding attempt $a$, where $n_{a+1} > n_{max}$.

Additionally, a success criterion may be tested during the decoding process. If the success criterion is satisfied, both the current decoding attempt and the complete decoding process are terminated. Hence, the success criterion is a stopping criterion

for both the decoding attempt and the decoding process.

A common success criterion for a Fountain decoding attempt (and for BP decoding in general) is to observe if the message estimate $\hat{\mathbf{m}}$ satisfies the constraints imposed by the parity check nodes. That is, for all $j \in \mathcal{I}_{n_a}$ we verify if the equation $\hat{x}_j \oplus \bigoplus_{m_i \in \mathcal{N}(c_j)} \hat{m}_i = +1$ is satisfied, where $\hat{x}_j$ is obtained through a hard decision on the channel LLR $Z_j$. This criterion is known as a check sum satisfaction ratio (CSR) test and uses the metric

$$\text{CSR} = \frac{s^{(l)}}{n_a}, \tag{2.10}$$

where $s^{(l)}$ is the number of satisfied check nodes at iteration $l$ of the current decoding attempt, and $n_a$ is the total number of check nodes of the current decoding attempt. The test is satisfied when the CSR is greater than or equal to a user defined threshold. This threshold is generally chosen to be 1 for complete parity check satisfaction.

A stopping criterion for a decoding attempt that allows us to terminate before the maximum number of iterations and before success has been achieved is called an *early termination* method for the decoding attempt. Similarly, a stopping criterion that allows us to terminate before reaching the maximum number of attempts and iterations for a decoding process although success has not been achieved is an early termination method for the entire decoding process. Early termination methods are used to reduce decoding cost when it is believed that further decoding would result in negligible performance improvement.

## 2.6 Raptor Decoder

For a Raptor code, the decoding performed at each decoding attempt may be classified into two categories: tandem decoding and joint decoding. In *tandem decoding*, at each decoding attempt, decoding is performed first completely for the LT code. Afterwards, the obtained soft information (e.g., LLRs) on the bits of the intermediate codeword $\tilde{\mathbf{m}}$ are input to the decoder for the precode. Using this information, the decoder for the precode obtains a message estimate $\hat{\mathbf{m}}$.

In *joint decoding*, soft information is exchanged not only from the Fountain code to the precode, but also from the precode to the Fountain code. After $l_{LT}$ BP iterations are performed for the LT code, soft information in the form of LLRs is input to the decoder for the precode. The precode decoder then performs $l_{precode}$ iterations and returns soft information to the LT decoder. This process is repeated for a certain number of rounds, after which the precode outputs $\hat{\mathbf{m}}$.

The precode of a Raptor code is usually an LDPC code. Similar to Fountain codes, LDPC codes are also decoded using the BP algorithm on a sparse factor graph. Thus, a factor graph that represents both the LDPC precode and the LT code may be constructed. Either tandem or joint decoding may then be performed on this graph. An analysis of tandem decoding for Raptor codes may be found in [6]. Analyses of joint decoding for Raptor codes may be found in [20, 28].

## 2.7 Fountain Codes on the BEC

When decoding on the BEC, columns of the generator matrix that correspond to erased encoded bits are also erased. It is this fragment of the generator matrix which we use for the decoding graph. For the BEC, we are completely certain of the value of any *received* bit. The value of a received bit $y_j$ is equal to the value of the transmitted bit $x_j$ with probability 1. This certainty allows us to vastly simplify the BP algorithm used on each decoding attempt [29].

Let $n_a$ be the number of received encoded bits at attempt $a$. The decoding process starts with $a = 1$. We will set $n_1 = 1$, as the decoding process can run greedily as bits arrive. The steps of the decoding process for a codeword are:

1. **Initialization of decoding attempt**: Set the check node $c_{n_a}$ to the value of the encoded node $x_{n_a}$ it is connected to. Go to step 2.

2. **Existence of degree one check node**: If a check node $c_j$ (where $j \in \mathcal{I}_{n_a}$) exists that is connected to only one unrecovered message node $m_i$, go to step 4. Else, go to step 3.

3. **Failure of decoding attempt**: Declare current decoding attempt a failure, receive one more encoded bit, set $a = a + 1$, set $n_a = n_a + 1$, and go to step 1.

4. **Recovery of message bit**: Set $m_i$=$c_j$. Declare $m_i$ as recovered. If all message bits are now recovered, declare decoding success and exit decoding process, else go to step 5.

5. **Check node updates and graph modification**: Add $m_i$ modulo-2 to all

check nodes $c_{j'}$, where $j' \neq j$, that are connected to $m_i$. Remove all edges connected to the message node $m_i$. Go to step 2.

The Fountain decoding process for the BEC decodes one message bit at a time using a check node of degree one in the decoding graph. Check nodes that were originally of a degree greater than one may become of degree one since edges of the decoding graph are erased throughout the decoding process. If a decoding attempt fails, the decoder waits for one more bit and attempts decoding again on the modified graph from the previous attempt. In practice, the decoder may not continue to make decoding attempts indefinitely, rather the encoder may have a maximum number of bits $n_{max}$ that it is allowed to transmit.

Two possibilities exist for the failure of a decoding attempt. Decoding may fail if there is an unrecovered message bit that is not connected to any check node. Alternatively, decoding may fail if of the set of unrecovered message bits none is connected to a degree one check node. Both cases are examples of *stopping sets*.

**Definition 2.2 :** A stopping set is a subset $\mathcal{S}$ of the set of variable nodes, such that all neighbours of $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice [30].

Successful decoding relies on the degree distribution. The degree distribution must ensure that there are no message bits without any connections to check nodes, and that as the decoding process advances there is always a check node of degree one. However, it must do this as efficiently as possible. The average degree of an encoded bit times the number of received encoded bits $n$ at the end of a successful decoding process is the average number of operations it takes to recover the message. Hence,

the degree distribution should allow for a decoding success with as few received encoded bits as possible (low overhead), and the encoded bits should have as low an average degree as possible.

The optimal number of received encoded bits for the BEC is $k$. For decoding to be successful with $k$ received bits, the average degree of the encoded bits must be at least $\log(k)$ (a proof of this can be found in [18]). In [4], Luby showed that this can be achieved using the *ideal soliton distribution*. Unfortunately, this distribution is very fragile. In practice, any slight deviation from the expected behaviour results in a decoding failure. This was Luby's motivation for proposing the *robust soliton distribution* [4]. The robust soliton distribution can successfully decode when slightly more than $k$ encoded bits have been received, and encoding and decoding complexity scale in $\log(k)$ per information bit. We recall that LT codes are Fountain codes specified by a degree distribution which they use in their encoding process. Thus, a Fountain code that uses either the ideal or robust soliton distribution in the three-step process described in Section 2.4.1 to generate an encoded bit, is an LT code.

## 2.8 Raptor Codes on the BEC

In [5], Shokrollahi showed that linear time encoding and decoding could be achieved with a Raptor code over the BEC, by concatenating a *weakened* LT code with a precode. The weakened LT code has constant average degree, instead of an average degree that scales in $\log(k)$. Thus, not all of the $k'$ input bits will have connections to output bits and only a constant fraction of the $k'$ input bits may be recovered. However, this fraction is sufficient for the precode to recover the original $k$ bits.

Raptor codes on the BEC have a constant (although arbitrarily small) overhead due to the precode, whereas the overhead for LT codes on the BEC vanishes as the number of message bits $k$ grows. This is the price paid for the linear time decoder.

## 2.9 Fountain Codes on Noisy Channels

All received symbols output from a noisy channel contain some information. Thus, we cannot discard symbols at the decoder without repercussions [31, 32], as in the erasure channel, nor are we completely certain of the values of the symbols we receive. In comparison with the BEC, noisy channels have a complex BP decoding algorithm. Furthermore, since each decoding attempt is run over the original decoding graph, i.e., no edges are erased, we can no longer afford to run the decoding process greedily each time the decoder receives a new symbol. Doing so may be optimal in terms of overhead, but would have impractically high decoding complexity. Thus, for noisy channels there is a rate performance-complexity tradeoff in the amount of symbols we wait to receive between decoding attempts. To circumvent this, [32] proposes to perform a decoding attempt only if a metric on the amount of received information indicates that there is sufficient information for the task. Moreover, in [26] it is shown that the decoder may wait for a small number of symbols $T$ between decoding attempts (other than the first), as long as incremental decoding is used and a small maximum number of iterations $l_{max}$ in relation to $T$ is used on these attempts. Additionally, in [33] the distribution of decoding times (whether we are able to successfully decode at a certain time) is examined for block fading channels.

In addition to decoding complexity, another main issue for Fountain codes on noisy channels is their performance with varying channel conditions. In [6] a lower bound is derived on the number of encoded bits of degree-2 necessary for a Raptor code or an LT code using BP decoding to achieve capacity. For the BEC channel, this quantity is a constant independent of the channel parameter. This is not the case for noisy channels. Thus, there is no single degree distribution that is *universal* (i.e., allows the realized rate to approach the channel capacity arbitrarily closely independent of the channel parameter) for LT codes or Raptor codes. However, one can characterize the robustness of Fountain codes on noisy channels by evaluating the variation of average realized rate over a range of channel capacities. In [20], degree distributions are designed for Raptor codes for the BIAWGN channel that are robust to channel variation. In [34, 35], another solution is proposed that would permit a Fountain code to potentially approach capacity with varying channel conditions: to allow the degree distribution to vary as bits are generated.

As discussed for Fountain codes on the BEC in Section 2.7, the design of the degree distribution is closely linked to the decoding process. Since the messages used on the BP decoding algorithm and the algorithm itself are different for noisy channels, the degree distribution must be designed accordingly. Work on degree distribution designs for noisy channels can be found in [6, 20, 28, 36, 37].

# Chapter 3

# Trapping Sets in Fountain Codes

## 3.1 Background of Trapping Sets

Trapping sets were first described for LDPC codes. A low-density parity check code is a linear block code defined as the kernel of a sparse matrix $\mathbf{H}$, called parity-check matrix. A sequence of bits $\mathbf{x}$ is a valid codeword if and only if $\mathbf{x}\mathbf{H}^T = \mathbf{0}$. Thus, the code bits of an LDPC codeword must satisfy the set of parity-check equations specified by the rows of $\mathbf{H}$.

LDPC codes are generally decoded using belief propagation on a factor graph that represents the parity check equations specified by $\mathbf{H}$. In Figure 3.1 we show an example decoding graph for an LDPC code. The number $r$ of check nodes for an LDPC code is greater than or equal to $(n - k)$. The LDPC decoding graph has only one type of variable node: the encoded node. When decoding, we try to estimate the value of the encoded bits rather than the message bits.

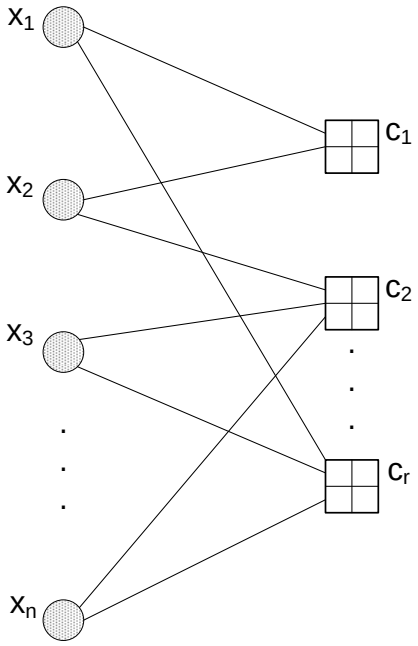



Figure 3.1: LDPC decoding graph.

For LDPC codes, the error performance curve usually has a slope that gracefully degrades for increasing signal to noise ratio (SNR) (i.e., for improving channel conditions), from low to moderate SNR and then has an abrupt decrease in the slope at high SNR. For this reason, the error performance curve is said to be divided into two regions: the waterfall region and the error floor region. The error floor is sometimes referred to as the bottom of the waterfall. The error floor problem has prevented the application of LDPC codes in many communication and storage systems which require an extremely low probability of error (bit error rates as low as $10^{-12}$ to $10^{-15}$ [38]) at code rates near the channel capacity.

The error floor is one of the most important open problems for LDPC codes. In [12], MacKay and Postol showed that the error floor for LDPC codes under belief

propagation decoding was primarily due to "near codewords". This concept was later generalized and titled as "trapping sets" by Richardson in [13]. Richardson chose the name trapping set to describe the subset of variable nodes that gets trapped on the incorrect values in the BP algorithm even though unsatisfied constraints (unsatisfied check nodes) remain. In this way, a trapping set prevents the decoder from converging to the correct message estimate.

The discovery of trapping sets on LDPC codes has led to a plethora of work in a variety of related subjects. Methods for error floor prediction were first described in [13] and subsequently by [39] and references therein. Techniques for error floor prediction are important since the time required for most computer Monte-Carlo simulations aimed at bit error rates of $10^{-9}$ and below is infeasible [40]. Work has also been realized to further the understanding of trapping sets [40–42]. Furthermore, research has been done on mitigating the error floor problem through trapping set knowledge.

Work on lowering the error floor can be divided into two main areas: code design and decoder design. For code design, the goal is to avoid certain topological structures in the LDPC decoding graph ([16] and references therein). Decoder-based strategies can be divided into two areas: decoder modification and post-processing [15]. The former changes the decoding algorithm used (e.g., scheduling of node message updates), while the latter uses the traditional algorithm, and once the decoder is caught in a trapping set applies a post-processing technique to correct the residual errors. Some decoder-based strategies for improving the error floor can be found in

[15, 43–46]. In what can be considered a decoder-based strategy, concatenating the LDPC code with a high rate precode can also be used to correct residual errors and lower the error floor [15].

**Definition 3.1 :** A $(w, z)$ trapping set $\mathcal{T}$ for belief propagation decoding of LDPC codes is a set of $w$ variable nodes whose induced subgraph in the decoding graph contains $z$ odd-degree check nodes [13]. A subgraph of the decoding graph is said to be induced if it has all the edges that appear in the decoding graph over the same set of nodes.

Not all inputs to the decoder will give rise to a failure event on a trapping set $\mathcal{T}$. When the input to the decoder gives rise to a failure event on a $(w, z)$ trapping set $\mathcal{T}$ at a given BP iteration, then the hard decision of the $w$ variable nodes is incorrect and the $z$ odd-degree check nodes in the induced subgraph are unsatisfied. We note that even-degree check nodes in the induced subgraph are satisfied, since an even number of errors still permits the parity check equation's constraint to be met.

From here on, when we state a variable node is erroneous or error-free, we are referring to the value of the hard decision on that variable node at a certain BP iteration. Additionally, we will refer to a failure event due to a trapping set as a *trapping set event*. Once a trapping set event occurs, we may classify it into three different categories (point, periodic, and aperiodic) [45] depending on the erroneous or error-free state of the variable nodes thereafter.

A *point* trapping set event is a subset of erroneous variable nodes that contains

all errors from a certain iteration up until the termination of the BP decoding algorithm. A *periodic* trapping set event is a subset of variable nodes whose values change periodically from erroneous to error-free. An *aperiodic* trapping set event is a subset of variable nodes who are in error at only one stage of the decoding algorithm.

## 3.2 Definition of Trapping Sets in Fountain codes

Similar to Fountain codes, LDPC codes are decoded using the belief propagation algorithm upon a factor graph. However, as discussed the decoding graph is structurally different. Moreover, they are fixed-rate codes. For these reasons, it was necessary to define and study trapping sets in Fountain in a different manner than that realized for LDPC codes.

LDPC codes have only one type of variable node, whereas Fountain codes have two types: message nodes and encoded nodes. It is important to make the distinction between the two types. In belief propagation decoding of Fountain codes we are interested in correctly estimating the value of the message nodes, not the encoded nodes. Furthermore, we define the check node degree as the number of connections it has to message nodes, only. Thus, we will describe a trapping set on a Fountain code with a triplet $(\alpha, \beta, z)$, where $\alpha$ is the number of message nodes in the trapping set, $\beta$ is the number of encoded nodes in the trapping set, and $z$ is the number of check nodes in the induced subgraph of the $\alpha$ and $\beta$ variable nodes in the trapping set. We let $\alpha+\beta = w$.

We define a trapping set for Fountain codes using the Fountain decoding matrix[1].

**Definition 3.2 :** For a given $u \times v$ matrix $\mathbf{O} = [o_{i,j}]$, the projection of a set of $r$ rows indexed by $i_1, i_2, \ldots, i_r$ is an $r \times v$ submatrix of $\mathbf{O}$ consisting of the elements $o_{i,j}$, where $i = i_1, i_2, \ldots, i_r$ and $j = 1, 2, \ldots, v$. A projection of a decoding matrix is a submatrix that is analogous to an induced subgraph of the decoding graph.

**Definition 3.3 :** Let $\mathbf{D}$ be a decoding matrix of a Fountain code. An $(\alpha, \beta, z)$ trapping set $\mathcal{T}$ is a set of $w = \alpha + \beta$ rows of $\mathbf{D}$ with a projection that contains $z > 0$ odd-weight columns.

The fact that encoded nodes may conform part of a trapping set is of interest, especially since the values of these nodes are constant (their LLRs are constant) throughout the entire decoding process. In contrast, the values of message nodes are subject to change at each decoding iteration and may therefore leave an erroneous state for an error-free state.

Furthermore, under the $(\alpha, \beta, z)$ trapping set definition we observe that a Fountain decoding matrix may contain a special subclass of trapping sets with $\alpha = 0$, $\beta > 0$, and $z > 0$. This case is noteworthy for having absolutely no message nodes in error and yet still having unsatisfied constraints. Thus, if a failure event occurs on a $(0, \beta, z)$ trapping set and our only stopping criterion is that all constraint nodes be satisfied, we will continue with the decoding process even though we have the correct message estimate[2]. For this subclass of trapping sets, $\beta$ is always equal to $z$, since

[1] A definition of trapping sets for LDPC codes based on the LDPC decoding matrix can be found in [46].

[2] Such a scenario does not occur for LDPC codes or for Fountain codes over the BEC.

the only variable nodes in error are encoded nodes. The projection of $\beta$ rows in an identity matrix will contain $\beta$ odd-weight (weight one) columns.

As examples, in Figures 3.2 and 3.3 we have marked in bold the induced subgraph of two trapping sets. The trapping set in Figure 3.2 consists of the encoded node $x_2$. The projection of $x_2$ in the decoding matrix has one column of weight one. This is described as a (0,1,1) trapping set. The trapping set in Figure 3.3 consists of three message nodes ($m_1, m_2$ and $m_k$) and two encoded nodes ($x_2$ and $x_{n_a}$). Additionally, the induced subgraph has three check nodes ($c_1$, $c_2$, and $c_3$) which are connected an odd number of times to the trapping set. The trapping set of Figure 3.3 is described accordingly as a (2,3,2) trapping set. If a failure event occurs on the trapping set of Figure 3.3, the variable nodes $m_1, m_2$, $m_k$, $x_2$ and $x_{n_a}$ will be in error and the check nodes $c_1, c_2$ and $c_3$ will be unsatisfied. Check node $c_{n_a}$ of the induced subgraph will be satisfied (or mis-satisfied), since it is connected to an even number of errors.

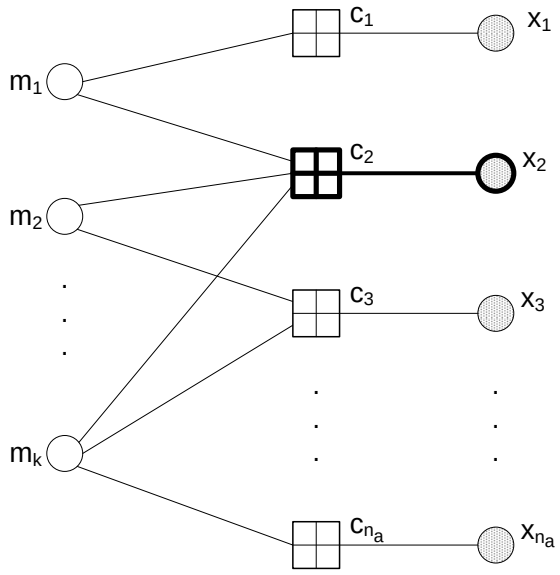



Figure 3.2: A (0,1,1) trapping set.

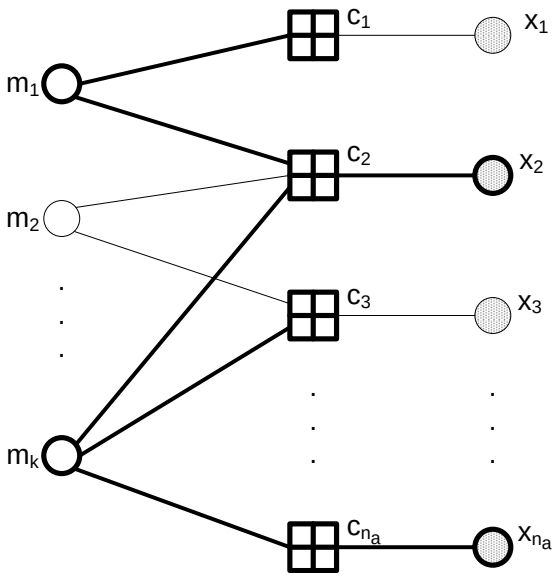



Figure 3.3: A (2,3,2) trapping set.

## 3.3 Objectives of Study

We have given a formal definition of trapping sets in Fountain codes. This definition was necessary to initiate our study on the subject. We then sought to obtain further insight through Monte Carlo simulation. These simulations aimed not only to collect information on the behaviour of trapping sets in Fountain codes, but also to collect information useful to the design of more efficient decoding schemes through trapping set knowledge. Below we discuss the main issues we aimed to address through our study.

**Decoding Failure due to Trapping Sets :** We aim to demonstrate through simulations that the failure of decoding attempts and processes on Fountain codes

may occur due to trapping set events.

**Point Trapping Sets :** While there are three main categories of trapping set events (point, periodic, aperiodic), our work will focus on point trapping set events only. Specifically, we investigate the behaviour and frequency of occurrence of two different types of point trapping set events: $(\alpha, \beta, z)$ point trapping set events with $\alpha > 0$ and $(\alpha, \beta, z)$ point trapping set events with $\alpha = 0$. We will refer to the latter simply as $(0, \beta, \beta)$ trapping sets.

**Effect of Rateless Property on Trapping Sets :** The simulations were designed to acknowledge the rateless property of Fountain codes. A rateless decoder progressively accumulates symbols until that time in which successful decoding can take place. Hence, a new decoding attempt integrates new information with previously received information. For the issue of trapping set events, this blending of old and new leads to an interesting question regarding the continuity or disruption of a failure due to a trapping set across decoding attempts. In the study, we observe whether the occurrence of a point trapping set event on one decoding attempt is indicative that a point trapping set event is likely to occur on the next decoding attempt, or whether the reception of more symbols will prevent this from happening. Additionally, we observe if, of two consecutive attempts with failures due to point trapping sets, one trapping set has members of the other trapping set. This is quantified through a measurement called the "trapping set survival ratio".

**Detection of Trapping Sets :** For point trapping set events, the number of variable nodes in error, $w$, and the number of unsatisfied check nodes, $z$, will remain constant from a certain decoding iteration onwards. In simulations, we can use our knowledge of the correct message to observe which variable nodes are in error and to which unsatisfied check nodes they are connected to in the decoding graph. However, in practice this is not possible. Nevertheless, we can use the observation that the check satisfaction ratio is stagnant to detect (with some probability of error) a point trapping set event. We say that the check satisfaction ratio is stagnant if $\Delta CSR = 0$ for a fixed number of iterations. Using the number of unsatisfied check nodes to identify trapping sets without knowledge of which variable nodes are erroneous has already been realized for LDPC codes in such works as [44, 46]. In our study, we observe the accuracy of this form of trapping set event detection.

**Early Termination through Trapping Set Detection :** The detection of trapping set events while decoding can be used for the early termination of a decoding attempt or to find a way of departing from the trapping set event. Through our observations on the behaviour of trapping set events, we will propose an early termination method for decoding attempts and an early termination method for the decoding process, both of which use trapping set detection.

## 3.4 Measurements

For our simulation, a number of messages are encoded by a Fountain code and transmitted over the channel. The received sequence per message block goes through the decoding process. Measurements taken during the simulation were chosen to address the aforementioned objectives of our study. Some measurements are taken per decoding iteration, others per decoding attempt, and yet others are collected for the entire simulation. Those collected for the entire simulation are accumulated across all realized decoding processes.

### 3.4.1 Per Decoding Iteration

- $\alpha$: The number of message bits in error. Also collected are the members in this set.
- $z$: The number of unsatisfied check nodes.
- CSR: Check satisfaction ratio.
- $\Delta$CSR: Difference in CSR between two consecutive iterations. Not collected for the first iteration of a decoding attempt.

### 3.4.2 Per Decoding Attempt

Each decoding attempt has only one value for each of the following measurements.

- $\beta$: The number of encoded nodes in error.
- success: Boolean variable that indicates whether the decoding attempt was successful or not.

- iter TS: Iteration at which point trapping set (TS) event (if any) began. This is obtained by comparing the members of the set of bits in error from a decoding iteration up to the end of the decoding attempt. They must be the same members from this iteration onwards for a point trapping set event to be identified.

- iter stagnant CSR: Iteration at which CSR becomes stagnant (if at all). $\Delta$CSR must have a value of 0 from this iteration onwards.

- Trapping set survival ratio: Let $V_a$ be the set of erroneous message nodes in a trapping set at attempt $a$. The trapping set survival ratio between two consecutive decoding attempts with trapping set events is

$$\frac{\mid V_a \bigcap V_{a-1} \mid}{\mid V_{a-1} \mid},$$

  where $a > 2$. We did not include the number of erroneous encoded nodes in the trapping set, since from a decoding perspective we are more interested in the set of erroneous message nodes. Additionally, an encoded node never changes its value. Let $E_a$ be the set of erroneous encoded nodes at attempt $a$, then for all $a$, $E_a \subseteq E_{a+1}$.

### 3.4.3 Across Simulations

These are measurements which have been accumulated across the span of the complete simulation.

- Number of completed decoding processes. This the same as the number of transmitted codewords, i.e., the same as the predetermined number of messages

to be encoded and transmitted in the simulation.

- Number of decoding attempts.
- Number of successful decoding attempts. This is the same as the number of successful decoding processes.
- Number of failed decoding attempts.
- Number of attempts with point TS events.
- Number of attempts with ($\alpha > 0$, $\beta$, $z$) point TS events.
- Number of attempts with (0, $\beta$, $\beta$) point TS events.
- Number of decoding attempts with stagnant CSR.
- Number of attempts where onset of TS and stagnant CSR coincide (iter TS and iter stagnant CSR must be within 5 iterations of each other).
- Number of attempts that follow an attempt which had an ($\alpha > 0$, $\beta$, $z$) point TS event.
- Number of attempts that follow an attempt which had a (0, $\beta$, $\beta$) point TS event.
- Number of successful attempts that follow an ($\alpha > 0$, $\beta$, $z$) point TS event.
- Number of successful attempts that follow a (0, $\beta$, $\beta$) point TS event.
- Number of attempts with point TS events that follow an attempt with a point TS event.

- Number of attempts that have a 90% or higher trapping set survival ratio.
- Average number of iterations before convergence to success in a decoding attempt.
- Average number of attempts before convergence to success in a decoding process.
- Average number of iterations before convergence to an ($\alpha > 0$, $\beta$, $z$) point TS event.
- Average number of iterations before convergence to a (0, $\beta$, $\beta$) point TS event.
- Total number of errors: measured by accumulating the number of errors at the end of each decoding process.
- Number of errors due to ($\alpha > 0$, $\beta$, $z$) point TS events: measured by accumulating the number of errors due to point TS events at the end of each decoding process.

## 3.5 Simulation Results

In the simulation, 30,000 message blocks were encoded using an LT code and transmitted over a BIAWGN channel with a value of $E_s/N_o$ set to 7.5 dB[3]. The number of message bits, $k$, in a message block was 1000 and the LT code used the degree distribution described in [18]:

[3]Less extensive simulations were also run for worse channel conditions, however a large overhead was necessary before the appearance of trapping sets. For this reason, we chose to present the simulation for 7.5 dB.

$$\Omega(v) = 0.007969v^1 + 0.493570v^2 + 0.166220v^3 + 0.072646v^4 + 0.082558v^5 \quad (3.1)$$
$$+ 0.056058v^8 + 0.037229v^9 + 0.055590v^{19} + 0.025023v^{65} + 0.003135v^{66}.$$

Traditional message reset decoding was used. The number of bits the receiver waited for before the first decoding attempt was 1000. The number of bits, $T$, the receiver waited for between decoding attempts was 100. The maximum allowable number of received symbols $n_{max}$ was 1600 (i.e., the maximum number of decoding attempts was 7). Per decoding attempt the maximum number of BP iterations, $l_{max}$, was 50.

The check satisfaction ratio was used to determine the success or failure of a decoding attempt. If CSR=1 at a given iteration of a decoding attempt, then both the decoding attempt and process were declared a success, and the decoding process was terminated. Otherwise, the decoding process continued. If the maximum number of iterations in a decoding attempt was reached without all constraints being satisfied, a decoding failure for the attempt was declared. If the maximum number of attempts and maximum number of iterations was realized for a decoding process, then the entire decoding process was declared a failure.

Table 3.1 contains a summary of the results for the simulation. The table contains the measurements which were accumulated across the entire simulation. The measurements per decoding iteration and per decoding attempt were mainly used to assist obtaining the measurements for the entire simulation.

Additionally, the measurements per decoding iteration were used to illustrate the behaviour of said measurements over the course of a decoding process. Figure 3.4 illustrates the behaviour of the number of message bits in error over the first three decoding attempts of a sample failed decoding process. Figures 3.5(a) to (c) illustrate the behaviour of three different measurements (number of message bits in error, CSR, and $\Delta$CSR) over all the decoding attempts of a sample failed decoding process. Figures 3.6(a) to (c) illustrate the behaviour of these same three measurements over the course of a sample successful decoding process.

The x-axis of each figure is divided into decoding attempts, these in turn are divided into decoding iterations. For Figures 3.4, 3.5(a) and 3.6(a) a red dashed line has been placed at the iteration corresponding to the onset of a point trapping set event within a decoding attempt. Next to the red dashed line we have placed the triplet $(\alpha, \beta, z)$ that characterizes the encountered trapping set event. In Figures 3.5(b) - (c) and 3.6(b) - (c), the red dashed line is placed at the iteration at which check satisfaction stagnation begins for a decoding attempt.

| **Name of Measurement** | **Value** |
|---|---|
| No. of completed decoding processes | 30,000 |
| No. of decoding attempts | 130,446 |
| No. of successful decoding attempts | 18,970 |
| No. of failed decoding attempts | 111,476 |
| No. of attempts with point TS events | 95,103 |
| No. of attempts with ($\alpha > 0$, $\beta$, $z$) point TS events | 49,312 |
| No. of attempts with (0, $\beta$, $\beta$) point TS events | 45,791 |
| No. of decoding attempts with stagnant CSR. | 90,409 |
| No. of attempts where onset of TS and stagnant CSR coincide | 90,398 |
| No. of attempts that follow an attempt which had an ($\alpha > 0$, $\beta$, $z$) point TS event | 48,825 |
| No. of attempts that follow an attempt which had a (0, $\beta$, $\beta$) point TS event | 35,248 |
| No. of successful attempts that follow an ($\alpha > 0$, $\beta$, $z$) point TS event | 11,462 |
| No. of successful attempts that follow a (0, $\beta$, $\beta$) point TS event | 0 |
| No. of attempts with point TS events that follow an attempt with a point TS event | 61,725 |
| No. of attempts that have a 90% or higher TS survival ratio | 49,088 |
| Avg. no. of iterations before convergence to success in a decoding attempt | 35.4086 |
| Avg. no. of attempts before convergence to success in a decoding process | 2.80633 |
| Avg. no. of iterations before convergence to an ($\alpha > 0$, $\beta$, $z$) point TS event | 20.3372 |
| Avg. no. of iterations before convergence to a (0, $\beta$, $\beta$) point TS event | 21.0581 |
| Total no. of errors | 15,991 |
| No. of errors due to ($\alpha > 0$, $\beta$, $z$) point TS events | 4,424 |

Table 3.1: Simulation results.

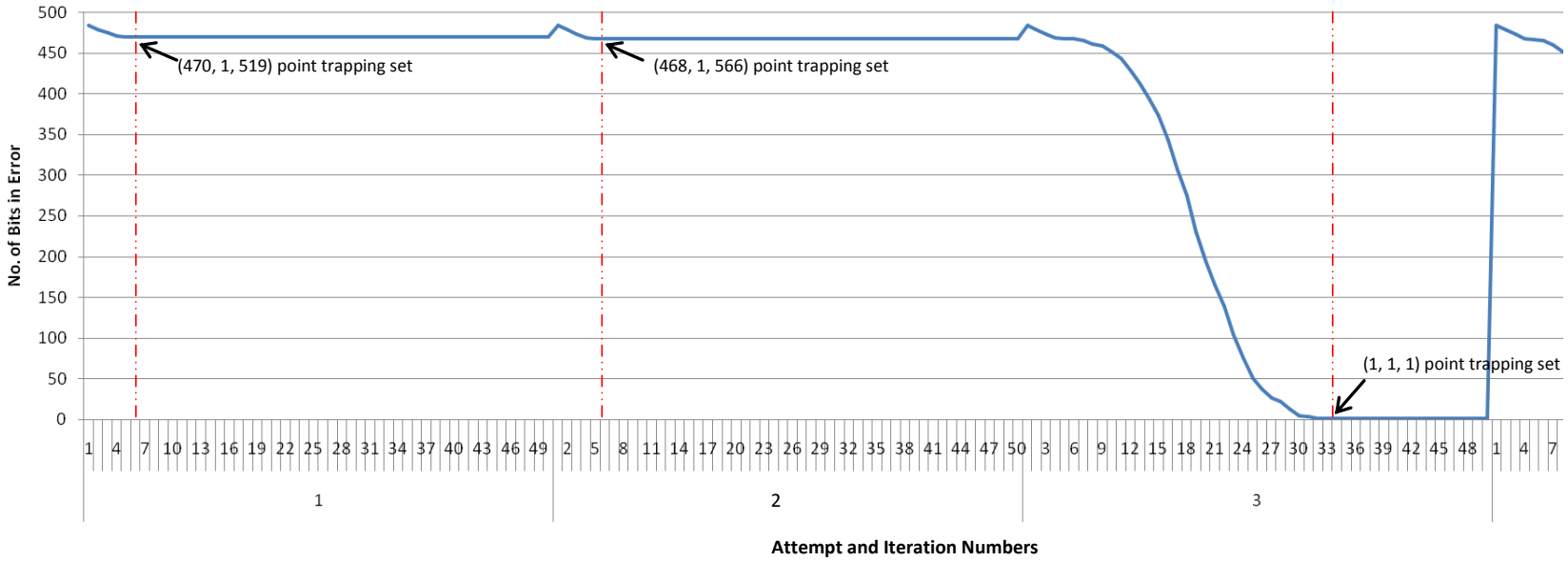
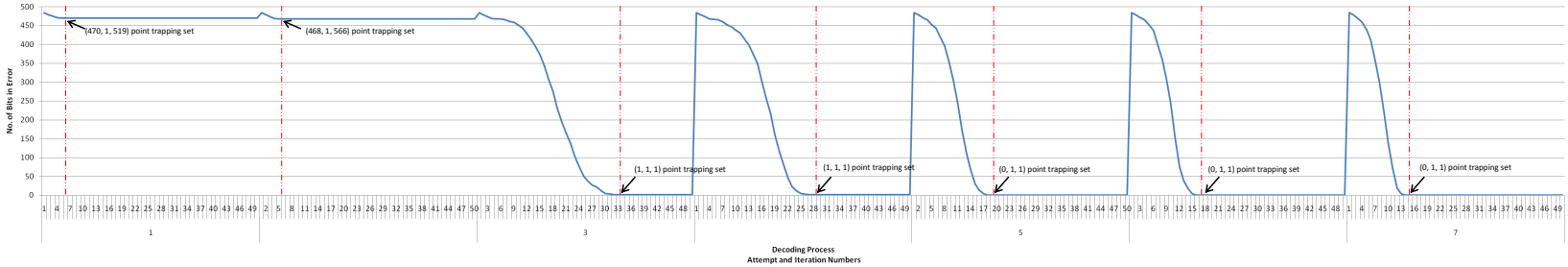
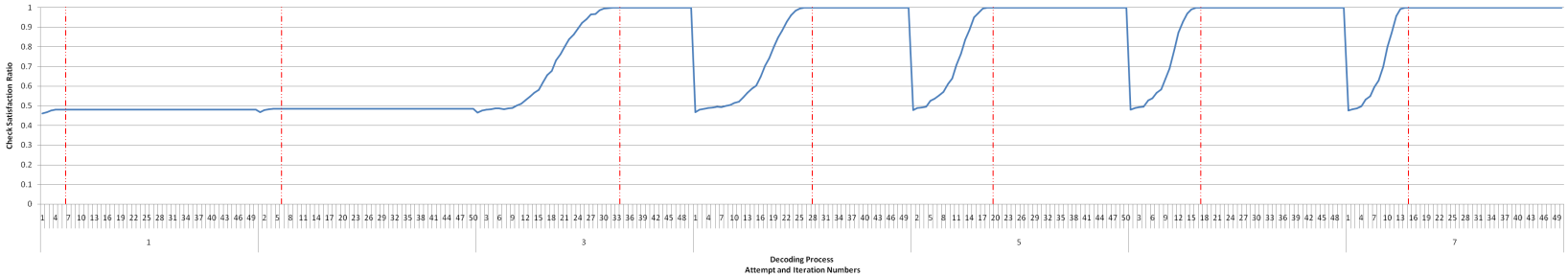


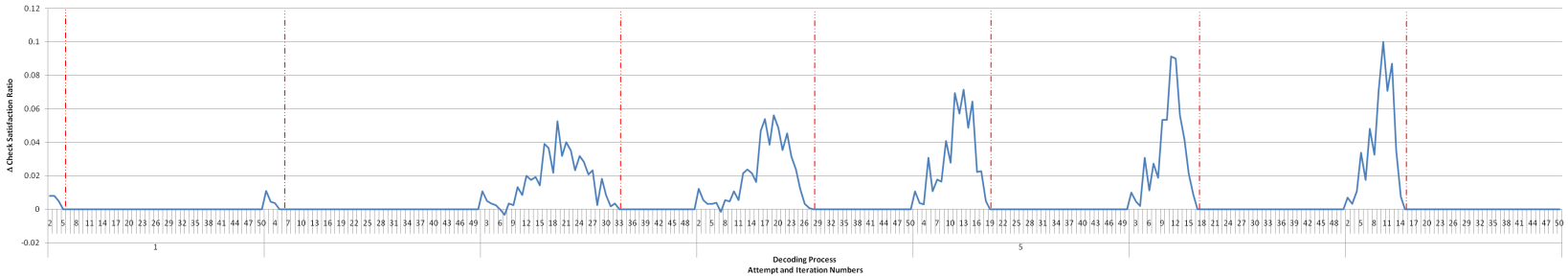

Figure 3.4: Message bits in error over the course of three decoding attempts. Red dashed line indicates onset of trapping set in a decoding attempt.




(a) Message bits in error

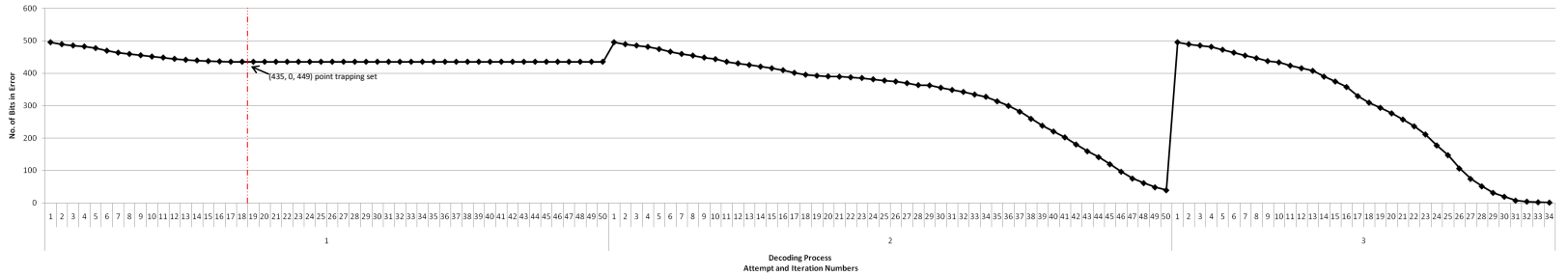



(b) CSR

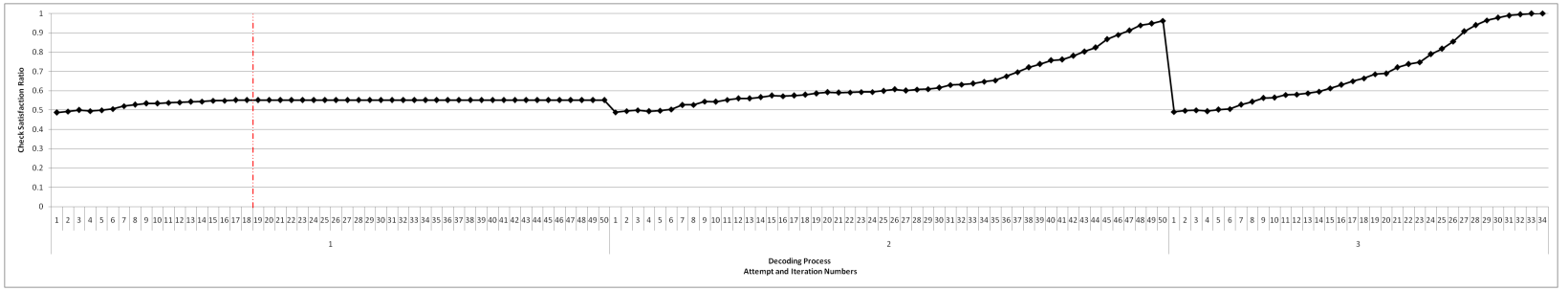



(c) ΔCSR

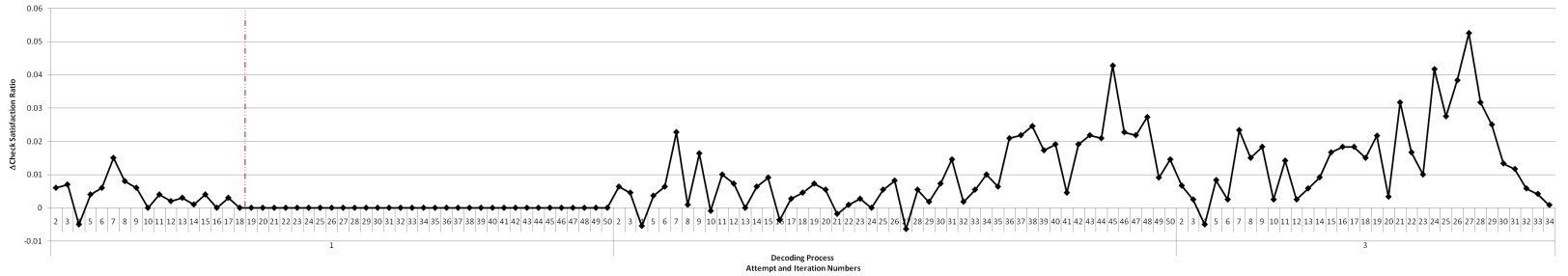
Figure 3.5: Measurements over the course of a sample failed decoding process. For figure (a) the red dashed line signals the onset of a trapping set in a decoding attempt. For figures (b) and (c) the red dashed line signals the onset of stagnant CSR in a decoding attempt.




(a) Message bits in error




(b) CSR




(c) ΔCSR

Figure 3.6: Measurements over the course of a sample successful decoding process. For figure (a) the red dashed line signals the onset of a trapping set in a decoding attempt. For figures (b) and (c) the red dashed line signals the onset of stagnant CSR in a decoding attempt.

## 3.6 Discussion of Results

### 3.6.1 On Decoding Failure and Error Performance

For belief propagation decoding, a decoding success is generally declared when all check node constraints are satisfied (i.e., CSR =1). This measure of success was used in the simulation. Under this definition, all point trapping set events cause decoding failures.

In the simulation there were 111,476 failed decoding attempts, 95,103 of these (around 85%) were due to point trapping sets. Thus, decoding failure events for decoding attempts were dominated by point trapping sets. Approximately 15% of the failed decoding attempts were due to other causes.

Only trapping sets with $\alpha > 0$ will cause errors in the message estimate. Approximately 28% of errors at the end of all decoding processes were due to ($\alpha > 0$, $\beta$, $z$), point trapping sets. Thus, the contribution of point trapping set events to the error performance of an LT code is worthy of attention.

If the decoder reports a decoding failure at the end of the decoding process, then any errors in the message estimate are called *detected errors*. If the decoder reports a decoding success at the end of the decoding process, then any errors in the message estimate are called *undetected errors*. Point trapping set events at the end of the decoding process cause a decoding failure. Hence, the fraction of errors contributed by point trapping set events to the detected errors of a code might be considerably

larger than that contributed to the total number of errors.

For Fountain codes which use complete parity check satisfaction as a success critierion, an undetected error is a message sequence $\hat{\mathbf{m}} \neq \mathbf{m}$ that satisfies $\hat{\mathbf{m}}\mathbf{G}' = \hat{\mathbf{x}}$. Since there is a one-to-one correspondence between $\mathbf{x}$ and $\mathbf{m}$, undetected errors occur when $\hat{\mathbf{x}} \neq \mathbf{x}$.

### 3.6.2 On the Effect of Ratelessness on Trapping Sets

Out of 30,000 transmitted codewords, 18,970 were successfully decoded. Of the 18,970 successful decoding processes, 11,462 contained one or more attempts with ($\alpha > 0$, $\beta$, $z$) point trapping set events. This demonstrates how the reception of more symbols may prevent an ($\alpha > 0$, $\beta$, $z$) point trapping set event from occurring on subsequent decoding attempts. In contrast, no successful decoding attempts followed a (0, $\beta$, $\beta$) point trapping set event. We also observed that a (0, $\beta_1$, $\beta_1$) trapping set was always followed by a (0, $\beta_2$, $\beta_2$) trapping set, where $\beta_2 \geq \beta_1$. Thus, in this case the reception of more symbols was of no benefit.

Almost 80% of attempts with trapping set events which were followed by an attempt that also had a trapping set had a trapping set survival ratio of 90% or more. Also, if a (0, $\beta_1$, $\beta_1$) trapping set was followed by a (0, $\beta_2$, $\beta_2$) trapping set (which in our simulation was always the case) then it had a trapping set survival ratio of 100%. Hence, it is frequent for a trapping set event to have a large number of members of the trapping set that caused a failure on the previous attempt.

### 3.6.3 On the Two Main Types of Point Trapping Sets

Within the failed decoding attempts due to point trapping set events around half of them were due to ($\alpha > 0$, $\beta$, $z$) trapping set events, the rest were due to (0, $\beta$, $\beta$) trapping set events.

Point trapping set events of the type $(0, \beta, \beta)$ contain encoded bit errors, but no message bit errors. For Fountain codes (and LDGM codes in general), we are only interested in estimating the message and not the codeword. Because of this, we will refer to $(0, \beta, \beta)$ trapping sets as 'error-free' trapping sets. Since these trapping set events are error-free, if we detect that one has occurred we may terminate the entire decoding process early (reduce decoding complexity and improve realized rate) at no expense in error performance. Otherwise, since attempts with error-free trapping set events are followed by other attempts with error-free trapping set events, then without early termination we will almost certainly incur the maximum expense in number of iterations and attempts. On the other hand, if we estimate that the trapping set is of type ($\alpha > 0$, $\beta$, $z$), then we may consider terminating the decoding attempt early, but awaiting the reception of more symbols for further attempts.

Additionally, if we are truly certain an error-free trapping set event has occurred, then we may tally this as successful decoding, regardless of the fact that not all constraints are satisfied.

### 3.6.4 On Trapping Set Detection

In the simulation, 90,409 decoding attempts showed stagnant check satisfaction, of these, 90,398 attempts had an onset of check stagnation which coincided (within 5 iterations) with that of the onset of a trapping set event. Since there are 95,103 attempts with trapping sets, we failed to detect some of the trapping set events using our measurement on stagnant check satisfaction. However, 95% of the time we accurately detected a trapping set within 5 iterations using the indicator of stagnant check satisfaction. In practice, due to complexity and delay considerations, it is likely that we would only observe if the check satisfaction is stagnant for several iterations (not until the maximum number of iterations) before declaring a trapping set event. Afterwards, we may either use post-processing to resolve the trapping set or perform early termination of the decoding attempt.

### 3.6.5 On the Behaviour of Measurements over the Course of a Decoding Process

Figure 3.5(a) displays the number of message bits in error at each stage of a sample failed decoding process. We observe that all decoding attempts ended with point trapping set events. The iteration at which a point trapping set event began has been signalled within each decoding attempt by a dashed line. Also, indicated is the triplet ($\alpha$, $\beta$, $z$) that describes the trapping set. The first two decoding attempts converged quickly (6 iterations) to trapping sets. Both trapping sets have high values of number of message bits in error and unsatisfied check nodes. In fact, the trapping set survival ratio between the first two decoding attempts is greater than 99%. With the reception of 100 more symbols in decoding attempt 3, a completely distinct trapping

set event surfaces with only 1 bit in error. The trapping set survival ratio between attempt 2 and 3 is 0%. However, in decoding attempt 4 the exact same (1, 1, 1) trapping set event from attempt 3 reoccurs, i.e., trapping set survival ratio is 100%. Decoding attempt 5 shows yet another trapping set event, this time it is an error-free (0, 1, 1) trapping set event, this error-free trapping set reappears in attempts 6 and 7. Thus, within 7 decoding attempts we observe 3 completely different trapping sets.

We observe that a specific trapping set event may 'survive' (i.e., have a substantial fraction of its members be part of the trapping set event on the next decoding attempt) or not occur again. Additionally, we note that the size $\alpha$ of the message node trapping set is larger at the beginning and decreases with more received symbols on subsequent decoding attempts. This is generally the case, with larger trapping sets appearing when the number of received symbols is less, and smaller or error-free trapping sets appearing once the number of received symbols is larger.

This sample decoding process from our simulation shows clearly how advantageous early termination based on the detection of trapping sets can be. If each decoding attempt had been terminated early at the iteration at which a trapping set began, then only 124 decoding iterations would have been completed compared to the 350 iterations realized without early termination[4]. Furthermore, if in addition to terminating each decoding attempt early, we had terminated the entire decoding process once the first error-free trapping set had occurred in attempt number 5, then we would have only performed 92 iterations. Moreover, the iterations we would have

[4]The amount of complexity reduction possible will depend on the selected parameters $(l_{max}, T, n_{max})$.

avoided performing from attempts number 6 and 7 are more expensive than those of previous attempts, since the number of received symbols is larger, and hence so are the number of edges in the decoding graph on which BP iterations are performed. Additionally, by not realizing attempts number 6 and 7 we would have improved our realized rate with no performance loss.

Figures 3.5(b) and 3.5(c) display the behaviour of the CSR and $\Delta$CSR over the course of the same sample decoding process of Figure 3.5(a). We have included them since ultimately these will be the measurements at our disposal to detect a trapping set event. For these figures the dashed line indicates the iteration at which check satisfaction stagnation begins. For this sample decoding process, the iterations at which check satisfaction stagnation begin match perfectly to the iterations at which the trapping sets begin. However, in some cases the iteration at which check satisfaction stagnation begins may occur a bit earlier.

Figures 3.6(a) to (c) display the same measurements as Figures 3.5(a) to (c), except now they are for a sample successful decoding process of our experiment. The first attempt ends in a rather large trapping set event. However, with the reception of more symbols on the second decoding attempt it does not reoccur. Finally, on the third attempt at the 34th iteration all check nodes are satisfied, successful decoding is declared and the decoding process terminates.

### Trapping Set Behaviour with Increasing Number of Attempts

As mentioned, trapping sets of the type $(\alpha > 0, \beta, z)$ may be resolved (not survive) with the reception of more symbols on subsequent attempts. Moreover, the value of $\alpha$ on subsequent attempts usually decreases. Thus, the reception of more symbols allows some of the members of the trapping set to overcome their bias towards the incorrect value. In [46] they discuss using redundant parity check equations (redundant check nodes) for LDPC codes to defeat a trapping set. Furthermore, they show that not all additional parity check equations are helpful in defeating a trapping set, some may even lead to other trapping sets. Rateless codes provide a natural solution for overcoming trapping sets, as a steady supply of new check nodes is always available. Additionally, a rateless code may be able to selectively choose which check nodes will provide the best result.

On the other hand, trapping sets of the type $(0, \beta, \beta)$ are unlikely to be resolved with the reception of more symbols on subsequent attempts. These trapping sets usually occur when the decoder has large $n$ or channel conditions are good or both. Thus, we believe that the bias of the message nodes in favour of the correct value is extremely strong. Moreover, the check satisfaction ratio at $(0, \beta, \beta)$ trapping set events is generally high. Thus, the majority of check nodes are reinforcing the correct value of the message nodes.

It is not desirable for the decoder to leave the $(\alpha = 0, \beta > 0, z = \beta)$ error-free trapping set. Since the number $\beta$ of erroneous encoded nodes is greater than 0, we have an incorrect $\hat{\mathbf{x}}$. Thus, the message estimate that satisfies the equation $\hat{\mathbf{m}}\mathbf{G}' = \hat{\mathbf{x}}$

will be incorrect. That is, we need the number $\alpha$ of erroneous message nodes to become greater than 0, so as to allow the number of unsatisfied check nodes $z$ to become 0. We would then have complete parity check satisfaction and declare a decoding success, however, we would no longer have the correct message estimate. This results in an undetected error.

### 3.6.6 More on Error-Free Trapping Sets

We have given the first description of error-free trapping sets. Specifically, we have described how this phenomenon may occur on Fountain codes. While we usually consider trapping sets as damaging to the error-performance of a code, we remark that convergence to an error-free trapping set is actually a positive occurrence. However, error-free trapping sets can be extremely damaging to the average decoding complexity and realized rate of a code. Under the traditional stopping criterion (all check sums must be satisfied) when an error-free trapping set event occurs, we fail to detect that we have arrived at the correct message estimate. This causes the decoder to continue performing unnecessary BP iterations and decoding attempts. This may happen several times given that it is likely that error-free trapping set events will reoccur on future decoding attempts.

It is interesting that the effects of error-free trapping sets had not been felt previously in the literature on Fountain codes. There are three reasons for this. First, in papers such as [6, 7] the BER and WER error performance of LT codes is explored at different fixed rates. For each fixed rate, the receiver buffers the corresponding $n$ symbols and performs BP decoding. Thus, if an error-free trapping set occurs the

decoder's complexity and realized rate are not worsened by realizing more decoding attempts as would happen in practice.

Second, in works such as [11, 32], where more than one decoding attempt is possible, the decision of either waiting for more symbols and performing a new decoding attempt or terminating the entire decoding process is determined by a cyclic redundancy check (CRC) instead of using the CSR. Because CRC is used, at the end of a decoding attempt where there is an error-free trapping set event it is likely that the entire decoding process will be stopped. This comes at the expense of added redundancy bits.

Third, much of the work on Fountain codes over noisy channels has been done for Raptor codes [6, 26], which are precoded LT codes. Generally, the LT decoding process is performed initially for a *fixed* number of iterations and then decoding is realized for the precode. The LT decoder provides as input to the decoder the calculated LLRs for the input bits. With an error-free trapping set, the hard-decision of these LLRs will be correct (a valid length $k'$ codeword sequence) and successful decoding is likely to occur while decoding for the precode. Thus, terminating the entire decoding process. This comes at the expense of added redundancy bits and a two-phase decoding process.

## 3.7 Recommendations

- The study of rateless codes using different fixed rates as realized in [6, 7] provides us with the performance of these codes for different block lengths. However,

to study the effect of specific properties of a rateless code we recommend each decoding process be allowed to take its natural course. In practice, each decoding process consists of several decoding attempts and the realized rate is not known a priori. Studying trapping sets for rateless codes in this way allowed us to observe their negative effect on realized rate. Specifically, trapping set events result in the failure of a decoding attempt. Thus, the decoding process continues, with each subsequent decoding attempt decreasing the realized rate. Additionally, studying trapping sets in a rateless setting led us to define the trapping set survival ratio. The trapping set survival ratio signals whether a trapping set event "survives" into the following decoding attempt.

- We recommend that trapping set detection on Fountain codes be used either for early termination or for post-processing of residual errors. Furthermore, we recommend that trapping set detection for Fountain codes classify trapping sets into two categories: trapping sets with errors and error-free trapping sets. The action taken after a trapping set event has occurred should depend on which category the trapping set falls into.

- If we detect a point trapping set event with errors, then early termination for the decoding attempt should be applied. Further iterations will not improve the message estimate. After early termination is applied, we should await the reception of more symbols for the next decoding attempt to improve the message estimate. Alternatively, when a point trapping set event with errors has occurred we may use a post-processing technique to correct residual errors.

- If we detect an error-free trapping set, then the entire decoding process should

be terminated. Since an error-free trapping set indicates that we have the correct message estimate, we should declare decoding success.

## 3.8 Trapping Set Detection

For point trapping set events, the set of variable nodes in error and the set of unsatisfied check nodes remains the same from a certain decoding iteration onwards. Thus, a detection method for point trapping set events is to observe if the check satisfaction ratio is stagnant, that is, if $\Delta\text{CSR} = 0$ for a fixed number $L$ of consecutive iterations.

Based on our observations, error-free point trapping sets usually occur when the CSR is very high and reoccur on subsequent decoding attempts. Whereas, point trapping sets with errors generally occur when the CSR is in the medium to high range and are resolved after more decoding attempts. Once we have detected a point trapping set has occurred, we may use other information available at the decoder to classify the trapping set as error-free or with errors. Thus, if the CSR is very high at the time a trapping set event occurs, we say it is an error-free trapping set. Otherwise we classify it as a trapping set with errors.

Alternatively, we may wait to see if several consecutive decoding attempts end in trapping set events at high values of CSR before we classify the trapping set event as error-free.

## 3.9 Early Termination Methods

Trapping set detection may be used for the early termination of decoding attempts and decoding processes. In this manner, improvement of decoding complexity and even realized rate are possible with negligible performance loss. We propose early termination methods based on trapping set detection to be applied jointly with any stopping criteria a particular Fountain decoding scheme may already have.

### 3.9.1 Early Termination for Decoding Attempts

Early termination of a decoding attempt is realized if a trapping set is detected. For the early termination of the decoding attempt it is not necessary to know if the trapping set is with errors or error-free. This is only necessary afterwards to decide whether we should await the reception of more symbols or not.

In [10, 11] they terminate a decoding attempt at the first instance in which $\Delta$CSR $= 0$. However, this may be only a false alarm and further decoding iterations may lead to a successful decoding attempt. Thus, if early termination is realized for this case, performance loss may be observed. We seek to reduce the possibility of a false assumption of convergence to a trapping set by only using early termination when $\Delta$CSR has been observed to have a value of 0 over $L > 1$ iterations, instead of just over a single iteration.

### 3.9.2 Early Termination for the Decoding Process

We wish to terminate the complete decoding process early if an error-free trapping set occurs. For this, if for $A$ consecutive decoding attempts the decoder finds itself in a trapping set ($\Delta$CSR= 0 for $L$ iterations) where the check satisfaction ratio is relatively high at the onset of the trapping set, then early termination of the decoding process is applied. Termination of the decoding process at an attempt with an error-free trapping set results in reduced decoding complexity and improved realized rate with no performance loss.

### 3.9.3 Steps for Early Termination

In the following steps we summarize the proposed actions for applying early termination if a trapping set is detected:

1. **Detect trapping set.** Done by verifying if $\Delta$CSR=0 over $L$ iterations.
2. **Apply early termination to the decoding attempt.**
3. **Classify trapping set.**
   (a) If CSR is low, trapping set is classified as being with errors.
   (b) If CSR is high, trapping set is classified as potentially error-free.
4. **Realize appropriate action**
   (a) If trapping set is with errors, then perform further decoding attempts.
   (b) If trapping set is potentially error-free, then verify if $A$ previous attempts also had trapping sets with high CSR.

i. If true, then declare error-free trapping set and terminate entire decoding process.

ii. If false, then perform further decoding attempts.

### 3.9.4 Pseudocode

The following pseudocode for a Fountain decoder using belief propagation integrates the steps mentioned in section 3.9.3.

**Pseudocode for Fountain decoder with trapping set detection and early termination**

```
1: n = 0 // current number of received symbols
2: n_a = n_1 // number of received symbols needed to perform the next decoding attempt
3: TShighCSR = 0 // number of consecutive decoding attempts that ended in a point trapping
   set with high CSR
4: process_stop = 0 // true when stopping criterion for decoding process is satisfied
   // Decoding Process
5: while process_stop == 0 do
6:       Collect n_a − n symbols
7:       for j = n + 1 to n_a do
8:            Get Z_j // Channel information of received symbol.
9:       end for
10:      n = n_a
11:      n_a = n + T
         // Decoding Attempt
12:      attempt_stop = 0 // true when stopping criterion for decoding attempt is satisfied
13:      l = 0 // initialize number of iterations performed
14:      l_stagnant = 0 // number of consecutive iterations with ΔCSR = 0
15:      Initialize M^(l)_{m_i→c_j} messages.
16:      while attempt_stop == 0 do
17:           Calculate all M^(l)_{c_j→m_i} according to equation (2.8).
18:           Calculate CSR
19:           Calculate m̂
20:           if m̂ satisfies success criterion then
21:                attempt_stop = 1
22:                process_stop = 1
23:           end if
24:           if l > 1 then
25:                Calculate ΔCSR
26:                if ΔCSR == 0 then
27:                     l_stagnant = l_stagnant + 1
28:                     if l_stagnant == L then // If a trapping set is detected
29:                          attempt_stop = 1 // Terminate decoding attempt
```

```
30:                             if CSR> 0.95 then
31:                                 TShighCSR = TShighCSR + 1
32:                                 if TShighCSR == A then // If trapping set is error-free
33:                                     process_stop = 1 // Terminate decoding process
34:                                 end if
35:                             else
36:                                 TShighCSR = 0
37:                             end if
38:                         end if
39:                     else
40:                         l_stagnant = 0
41:                     end if
42:                 end if
43:                 l = l + 1
44:                 if l == l_max then
45:                     attempt_stop = 1
46:                     if n_a > n_max then
47:                         process_stop = 1
48:                     end if
49:                 end if
50:                 Calculate all M^(l)_{m_i→c_j} according to equation (2.9).
51:         end while
52: end while
```

# Chapter 4

# Conclusions and Future Work

## 4.1 Conclusions

In this work, we formally defined trapping sets in Fountain codes over noisy channels. This was necessary due to the distinct properties of Fountain codes. Additionally, we gave the first definition of an error-free trapping set. Furthermore, we realized an experimental simulation to show the behaviour of trapping sets in the rateless Fountain code setting. These experiments demonstrated the importance of trapping sets on Fountain codes. For instance, we showed that trapping sets occur while decoding on Fountain codes, thus causing decoding failure. Additionally, we observed how detecting trapping sets can be a means to improve both decoding complexity and realized rate with negligible performance loss. We also witnessed how ($\alpha > 0$, $\beta$, $z$) trapping sets affect the error performance of a Fountain code.

Motivated by the rateless property of Fountain codes, we are the first to define the trapping set survival ratio and show how a trapping set may reappear as part of

another trapping set in future decoding attempts or be defeated by the reception of more symbols. In particular, for our simulation we saw that the trapping set survival ratio of an error-free trapping set was always 100%. We also provided a detailed discussion on the phenomenon of error-free trapping sets. Finally, using the knowledge obtained from our experimental simulation we proposed an early termination method for decoding attempts and one for the complete decoding process.

In this work, we focus on the problem of trapping sets encountered during the decoding process of Fountain codes. Our results, however, are applicable to Raptor codes and the general class of precoded Fountain codes. Both tandem and joint decoding of a precoded Fountain code need to perform decoding on a Fountain code. Furthermore, our results (excluding those regarding ratelessness) are applicable to fixed-rate LDGM codes.

## 4.2 Future Work

- Apply proposed early termination methods and realize a performance-complexity analysis that compares our methods with those existing in the literature.
- Perform simulations that investigate trapping set behaviour for different flavours of BP decoding for Fountain codes (e.g., incremental decoding), different parameter settings, different channels, and different channel conditions.
- For precoded Fountain codes (e.g., Raptor codes), study the effect of the precode decoding process on the trapping sets of the Fountain code. Additionally, if the precode also has trapping sets (e.g., LDPC codes), study the effect of the

Fountain decoding process on these.

- Use rich literature on trapping sets in LDPC codes to better understand trapping sets in Fountain codes and design codes/decoders that either avoid or combat trapping sets. While much of the literature for LDPC codes makes use of knowing the code beforehand, the rateless property of Fountain codes is certain to provide interesting solutions to these design problems.

# Bibliography


[1] D. Mandelbaum, "An adaptive-feedback coding scheme using incremental redundancy," *IEEE Trans. Inf. Theory*, vol. 20, no. 3, pp. 388–389, May 1974.

[2] S. Sesia, G. Caire, and G. Vivier, "Incremental redundancy hybrid ARQ schemes based on low-density parity-check codes," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1311–1321, Aug. 2004.

[3] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389–400, Apr. 1988.

[4] M. Luby, "LT codes," in *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 271–280.

[5] A. Shokrollahi, "Raptor codes," in *Proc. IEEE International Symposium on Information Theory*, Jun./Jul. 2004, p. 36.

[6] O. Etesami and A. Shokrollahi, "Raptor codes on binary memoryless symmetric channels," *IEEE Trans. Inf. Theory*, vol. 52, no. 5, pp. 2033–2051, May 2006.

[7] R. Palanki and J. S. Yedidia, "Rateless codes on noisy channels," Tech. Rep., 2004. [Online]. Available: www.merl.com/papers/TR2003-124/

[8] J. Castura and Y. Mao, "Rateless coding over fading channels," *IEEE Commun. Lett.*, vol. 10, no. 1, pp. 46–48, Jan. 2006.

[9] Y. Cao and S. Blostein, "Raptor codes for hybrid error-erasure channels with memory," in *Proc. 24th Biennial Symposium on Communications*, Jun. 2008, pp. 84–88.

[10] A. AbdulHussein, A. Oka, and L. Lampe, "Decoding with early termination for Raptor codes," *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 444–446, Jun. 2008.

[11] ——, "Decoding with early termination for rateless (Luby Transform) codes," in *Proc. IEEE Wireless Communications and Networking Conference*, Mar./Apr. 2008, pp. 249–254.

[12] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[13] T. Richardson, "Error floors of LDPC codes," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003, pp. 1426–1435.

[14] S. Laendner and O. Milenkovic, "LDPC codes based on latin squares: Cycle structure, stopping set, and trapping set analysis," *IEEE Trans. Commun.*, vol. 55, no. 2, pp. 303–312, Feb. 2007.

[15] Y. Han and W. Ryan, "LDPC decoder strategies for achieving low error floors," in *Proc. Information Theory and Applications Workshop*, Mar./Feb. 2008, pp. 277–286.

[16] M. Ivkovic, S. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug. 2008.

[17] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, Mar. 2008.

[18] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.

[19] E. Hyytia, T. Tirronen, and J. Virtamo, "Optimal degree distribution for LT codes with small message length," in *Proc. 26th IEEE International Conference on Computer Communications*, May 2007, pp. 2576–2580.

[20] A. Venkiah, C. Poulliat, and D. Declercq, "Jointly decoded Raptor codes: analysis and design for the BIAWGN channel," *EURASIP Journal on Wireless Communications and Networking*, 2009.

[21] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[22] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[23] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/itprnn/book.html

[24] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Feb. 2001.

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

[26] K. Hu, J. Castura, and Y. Mao, "Reduced-complexity decoding of Raptor codes over fading channels," in *Proc. IEEE Global Telecommunications Conference*, Nov. 2006, pp. 1–5.

[27] ——, "Performance-complexity tradeoffs of Raptor codes over Gaussian channels," *IEEE Commun. Lett.*, vol. 11, no. 4, pp. 343–345, Apr. 2007.

[28] A. Venkiah, C. Poulliat, and D. Declercq, "Analysis and design of Raptor codes for joint decoding using information content evolution," in *Proc. IEEE International Symposium on Information Theory*, Jun. 2007, pp. 421–425.

[29] D. MacKay, "Fountain codes," *IEE Proc. - Comm.*, vol. 152, no. 6, pp. 1062–1068, Dec. 2005.

[30] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes," *IEEE Trans. Inf. Theory*, pp. 1570–1579, Jun. 2002.

[31] D. Park and S.-Y. Chung, "Performance-complexity tradeoffs of rateless codes," in *Proc. IEEE International Symposium on Information Theory*, Jul. 2008, pp. 2056–2060.

[32] H. Jenkac, T. Mayer, T. Stockhammer, and W. Xu, "Soft decoding of LT-codes for wireless broadcast," in *Proc. IST Mobile Summit*, Dresden, Jun. 2005.

[33] J. Castura and Y. Mao, "When is a message decodable over fading channels?" in *Proc. 23rd Biennial Symposium on Communications*, Kingston, Ontario, May 2006.

[34] H. Pishro-Nik and F. Fekri, "On Raptor codes," in *Proc. IEEE International Conference on Communications*, vol. 3, Jun. 2006, pp. 1137–1141.

[35] N. Bonello, R. Zhang, S. Chen, and L. Hanzo, "Reconfigurable rateless codes," in *Proc. VTC Spring*, Barcelona, Spain, Apr. 2009.

[36] P. Pakzad and A. Shokrollahi, "Design principles for Raptor codes," in *Proc. IEEE Information Theory Workshop*, Punta del Este, Mar. 2006, pp. 165–169.

[37] Z. Cheng, J. Castura, and Y. Mao, "On the design of Raptor codes for binary-input Gaussian channels," in *Proc. IEEE International Symposium on Information Theory*, Jun. 2007, pp. 426–430.

[38] X. Hu, B. Vijaya Kumar, Z. Li, and R. Barndt, "Error floor estimation of long LDPC codes on partial response channels," in *Proc. IEEE Global Telecommunications Conference*, Nov. 2007, pp. 259–264.

[39] P. Lee, L. Dolecek, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Error floors in LDPC codes: Fast simulation, bounds and hardware emulation," in *Proc. IEEE International Symposium on Information Theory*, Jul. 2008, pp. 444–448.

[40] M. Stepanov and M. Chertkov, "Instanton analysis of low-density parity-check codes in the error-floor regime," in *Proc. IEEE International Symposium on Information Theory*, Seattle, Jul. 2006, pp. 552–556.

[41] C.-C. Wang, "On the exhaustion and elimination of trapping sets: Algorithms & the suppressing effect," in *Proc. IEEE International Symposium on Information Theory*, Jun. 2007, pp. 2271–2275.

[42] R. Koetter and P. Vontobel, "Graph covers and iterative decoding of finite-length codes," in *Proc. 3rd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sep. 2003, pp. 75–82.

[43] A. Vila Casado, M. Griot, and R. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proc. IEEE International Conference on Communications*, Jun. 2007, pp. 932–937.

[44] E. Cavus and B. Daneshrad, "A performance improvement and error floor avoidance technique for belief propagation decoding of LDPC codes," in *Proc. IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 4, Sep. 2005, pp. 2386–2390.

[45] S. Landner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *Proc. IEEE International Conference on Wireless Networks, Communications and Mobile Computing*, vol. 1, 2005, pp. 630–635.

[46] S. Laendner, T. Hehn, O. Milenkovic, and J. Huber, "When does one redundant parity-check equation matter?" in *Proc. IEEE Global Telecommunications Conference*, Nov./Dec. 2006, pp. 1–6.