uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ketai Hu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Reduced-Complexity Decoding of Fountain Codes

TITRE DE LA THÈSE / TITLE OF THESIS

Yongyi Mao

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

I. Marsland

C. D`Amours

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

REDUCED-COMPLEXITY DECODING OF FOUNTAIN CODES

KETAI HU

A Thesis submitted to the Faculty of Graduate Studies and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Master of Applied Science, Electrical Engineering

April 2006

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Informaiton Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-18427-1
Our file Notre référence
ISBN: 978-0-494-18427-1

To my parents

Reduced-Complexity Decoding of Fountain Codes

Master of Applied Science, Electrical Engineering Thesis
School of Information Technology and Engineering
University of Ottawa

by Ketai Hu
April 2006

Abstract

Fountain codes are a universal class of rateless codes originally designed for erasure channels. Naturally adapting to channel states without channel knowledge at the transmitter, Fountain codes have recently been demonstrated also as an appealing solution for communication over fading channels. However, their relatively high decoding complexity limits their practical use in a wireless setting. In this thesis, we present a new decoding algorithm for Raptor codes — a type of Fountain codes — over fading channels, where the complexity is significantly reduced without sacrifice of performance.

Acknowledgments

First, I would like to thank my supervisor, Professor Yongyi Mao, for his guidance, encouragement and support during my graduate studies. He has created a demanding educational environment combined with a social and celebratory spirit. While the skills and knowledge that I have acquired are certainly significant, I truly enjoyed myself along the way.

Next, I wish to express my deep gratitude to Jeff Castura, for his generous help to verify my algorithm and run the simulations over the channel with capacity 0.25 bits/channel use, for his insightful comments and suggestions. Without his verification, I would not have confidence to carry out the new algorithm.

All of our group have helped at various aspects of my study life at University of Ottawa. Especially, they selflessly provided their computers and other computing resource such as HPCVL accounts during the simulation period. I am grateful to all of them. Zhong has provided helpful suggestions on my algorithm occasionally, Xueying has assisted me in data processing. I apologize I can not mention everybody's name here.

Last, but not least, I would like to thank everyone else who has helped me to see that I have chosen the right path.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless communication has brought a surge of research activities in past decade. In this setting, it is often the case that the communication system needs to be designed without the transmitter's knowledge of channel states. Over fading channels with large coherence time, the challenge of channel uncertainty translates to an inevitable trade-off between the achievable rate and the outage probability.

The invention of Fountain codes [1, 2] — including LT codes and Raptor codes, originally designed for erasure channels, has signaled a new paradigm for communication under channel uncertainty [3, 4]. Without the need of channel knowledge and even channel statistics, Fountain codes have demonstrated to be universally capacity-achieving over erasure channels. This success has inspired great research interest recently in information-theoretic generalization of Fountain codes to "rateless codes" [5, 6] and the applications of these codes to other settings of channel uncertainty. In particular, Fountain codes have been used for communication over wireless channels and "universal-like" and capacity-approaching performance was shown [7].

However, unlike over erasure channels, the existing decoding algorithms for decoding, Fountain codes over Gaussian and fading channels incur relatively high complexity. This fact to an extent limits the practical use of Fountain codes for fading channels.

In this thesis, we propose a new algorithm, which is a simple modification of the decoding algorithm for Raptor codes — a type of Fountain codes — over fading channels. Similar to the original decoding algorithm, our algorithm is still based on the sum-product

algorithm [8] on the factor-graph representation of the code. But instead of resetting the decoder at every decoding attempt, the results of previous decoding attempt are used to initialize the current decoding attempt. We show that comparing with the existing decoding algorithms, the modified algorithm in this thesis offers a significantly reduced decoding complexity with no performance loss and even performance gain.

Specifically, the contribution of this thesis is as follows.

- A new decoding algorithm for Raptor codes over fading channels is proposed.
- Theoretical justification of the proposed algorithm is given.
- The algorithm is simulated over various channel settings and significant advantage in complexity and in performance-complexity trade-off is demonstrated.

This remainder of thesis is organized as follows.

Chapter 2 gives necessary background that is need for the development of this thesis. Chapter 3 examines the existing algorithm for decoding Raptor codes over fading channels, and outlines the performance and complexity metrics considered in this work, and proposes the new algorithm with theoretical justification. Chapter 4 presents simulation results. The thesis is briefly concluded in chapter 5.

Chapter 2

Background

2.1 Communication under channel uncertainty

2.1.1 The challenge

In many communication scenarios, the communication system needs to be designed without complete channel knowledge. This results in the challenge of communication under channel uncertainty [3]. One such example of great practical significance is the wireless communication setting, where for example, the transmitter, the receiver, or both, have no knowledge of the channel gain or even the channel statistics. A typical wireless channel — usually referred to as a *fading channel* — may be modeled in general as

$$y_i = h_i x_i + z_i \tag{2.1}$$

where, at time i, y_i and x_i are respectively the received and transmitted complex symbols, h_i is the channel gain, and z_i is the complex additive white Gaussian noise.

We note that here we only consider the point-to-point single-antenna communication setting, where h_i at each time instant i is a complex scalar. Due to the stochastic nature of wireless channels, $\{h_i : i = 1, 2, \ldots\}$ is in fact a random process. One of the most popular models for this process is known as the *block-fading* model, where h_i stays as a constant for every time block of duration T_c, and assumes independent values — drawn

from some distribution p_h — across blocks. These blocks are typically referred to as *coherence blocks* and T_c referred to as *coherence time* [9]. On one extreme, if $T_c = 1$, the channel may be called a *fast-fading channel.* On the other extreme, if T_c assumes a very large number — larger than the codeword length for which one may take $T_c = \infty$, the channel may be called a *slow-fading channel.* Of course, there exist a wide range of channel behaviors in between these two extremes.

The distribution p_h may be modeled in various forms. For the purpose of this work, the form of p_h has little impact on the results and conclusion of this research, as we will explain later.

Our study primarily deals with the communication setting over such channels in which the fading process $\{h_i : i = 1, 2, \ldots\}$ is known at the receiver but unknown at the transmitter. This will exclude the fast-fading scenarios from our consideration, as in those cases, significant difficulty exists in channel estimation and the assumption of receiver's knowledge of the fading process is unlikely to be realistic.

We now consider slow-fading channels and channels whose coherence time T_c is relatively large with respect to the length of the codewords. Such a scenario in reality corresponds to a situation where certain latency constraint forbids the use of very long codewords and a codeword spans *one or only a few* coherence blocks. In these cases, there is a well-known trade-off — at least with the conventional fixed-rate coding schemes — between communication *efficiency* and *reliability* [9], which will be sketched as follows.

For instance, consider a slow-fading channel, where channel gain h_i in (2.1) reduces to a single random variable h, independent of time. Given h, we may refer to

$$C(h) := \log_2(1 + \gamma |h|^2) \text{ bits/channel use}, \tag{2.2}$$

as the *realized capacity* of the channel, where γ is the transmit SNR. Suppose that over this channel, the transmission of a $k-$bit message is via a codeword of n symbols. Then we refer to

$$R := k/n \tag{2.3}$$

as the *realized rate* for transmitting this message. In the conventional fixed-rate coding scheme, the realized rate is fixed *a priori*, and when $R > C(h)$, an *outage* event —

decoding error or decoding failure — is inevitable to occur and the limiting probability of outage is given by

$$P_{out} = P[R > C(h)]. \tag{2.4}$$

In Figure 2.1, a realized-capacity distribution is sketched, and for a given realized rate R fixed *a priori*, the shaded area indicates the limiting outage probability.

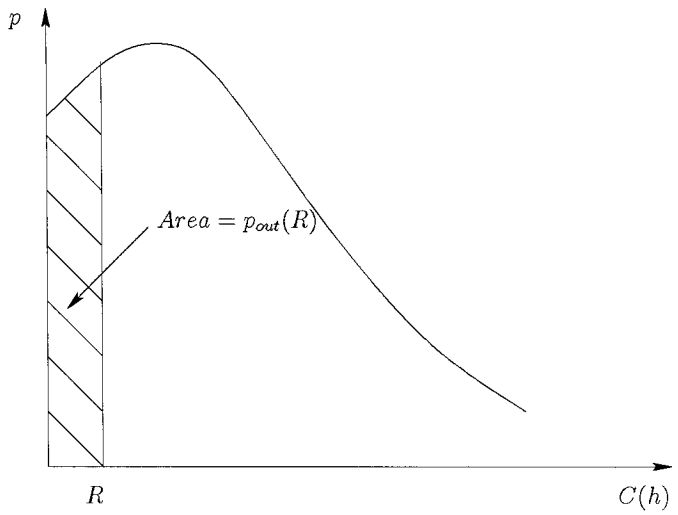

Figure 2.1: Probability density function of $C(h)$ for a fading channel, where the shaded area indicates the limiting outage probability.

Under most models of p_h, such as Rayleigh or Rician model, the tail of the realized-capacity distribution usually extends to infinity. This immediately implies that in order to reduce the outage probability, one must sacrifice the rate of communication, and the trade-off between efficiency and reliability is therefore seen.

As the coherence time T_c decreases, one expects that such a trade-off persists until T_c is very small comparing with the codeword length [1]. From the example of slow-fading outlined above, it is clear that such a trade-off is a consequence of the lack of channel knowledge at the transmitter and the communication rate can not be selected *a priori* to match the realized capacity.

[1] At the small T_c extreme or for the fast fading case, since a codeword may span many coherence blocks, the time average of the realized capacity over these blocks approaches the *ergodic capacity* [9] in probability, and it is possible to set the rate *a priori* to close to the ergodic capacity while maintaining arbitrary small outage probability.

2.1.2 The paradigm of rateless coding

Existing approaches to resolving this tension, such as an adaptive coding and modulation scheme, typically involve feeding back the channel state information (for example in the wireless setting given in (2.1), the estimate of h_i) to the transmitter. This may allow the transmitter to choose a code with rate better matching the realized capacity.

The paradigm of rateless coding, pioneered by the invention of *Fountain codes* [1,2] for erasure channels, has recently demonstrated as an efficient approach of using feedbacks for communication under channel uncertainty [5, 7]. Unlike the fixed-rate codes which encode a k-bit message to a length-n codeword for some pre-determined k and n (and hence a fixed rate k/n), in a rateless coding scheme, the transmitter encodes the k-bit message into a potentially infinite stream of symbols and starts transmitting the symbols over the channel. The receiver keeps attempting to decode as the symbols are received. When the decoder is sufficiently confident that it can decode, it declares an estimate of the transmitted message and sends an `ACK` to the transmitter via a feedback channel. The transmitter then terminates the transmission of this codeword and starts transmitting the next codeword if needed.

Over erasure channels, Fountain codes, including LT codes and Raptor codes, have demonstrated to be universally capacity-achieving, where channel knowledge is needed neither at the transmitter nor at the receiver [1, 2]. Recently, Fountain codes have also been tested over binary symmetric channels, AWGN channels, and fading channels, where capacity-approaching and "universal-like" performance are also seen [7, 10, 11]. In particular, the authors of [7] showed that Fountain codes are capable of achieving an averaged realized rate only within a small gap away from the averaged realized capacity.

In fact, the information-theoretic results of [5] have shown that over discrete memoryless channels, rateless codes can achieve the induced mutual information across the channel (by the input distribution selected by the transmitter) while neither the transmitter nor the receiver needs to know the channel states or even the channel law. In addition, the authors of [6] proved that there exist rateless codes, beyond the known constructions of Fountain codes, in which the realized rate over block fading channels can be made arbitrarily close to the realized capacity, and at the same time, the outage probability can be made arbitrarily small. That is, in principle, an optimal rateless code

can simultaneously achieve both efficiency and reliability, thereby resolving the trade-off inherent in the fixed-rate coding schemes.

2.2 Fountain codes

Fountain codes are originally designed for reliable communications over erasure channels. The idea of Fountain codes over erasure channels may be described using the following analogy. The transmitter is a fountain that produces an endless supply of water drops (encoded packets). The receiver is a cup with the objective of collecting k water drops. The fountain keeps spraying, and some water drops may get lost on their way. But as long as long as the cup collects k (in practice, slightly larger than k) drops of water, the objective is achieved.

Although this analogy is the original motivation for naming the codes "Fountain codes", the precise description of these codes and their decoding algorithms (particularly over fading channels) requires a careful introduction of the framework of factor graphs, the sum-product algorithm, and their applications in error control coding.

2.2.1 Factor graphs

Synthesized from the development of Tanner graphs [12] and Tanner-Wiberg-Loeliger graphs [13], a *factor graph* is a bi-partite graph consisting of two types of nodes, *variable nodes* and *function nodes*. The set of variable nodes each represent a variable, and the set of function nodes each represent a function, which is often referred to as a *local function*. A function node is connected to a variable node if and only if the variable is in the argument of the function. The graph represents the product of all the local functions, and this product is referred to as the *global function* represented by the factor graph. For example, the factor graph in Figure 2.2 represents the global function

$$f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_3, x_4, x_5)$$

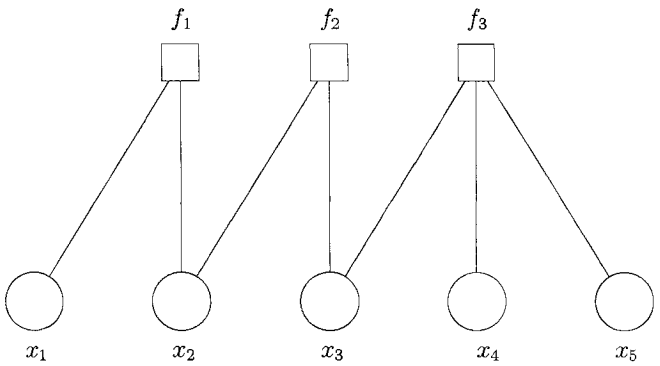

Figure 2.2: A factor graph representing product $f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_3, x_4, x_5)$, where squares represent functions nodes, and circles represent variable nodes.

2.2.2 The sum-product algorithm

Using a factor graph to represent a function that factors as a product of many local functions, an efficient algorithmic tool known as the sum-product algorithm can be used for simultaneously computing all "marginals" of the global function. We now give a precise description of the sum-product algorithm.

Let factor graph $\mathcal{G}$ represent the factorization of function

$$g(x_1, \ldots, x_n) := \prod_{j=1}^{m} f_j(X_j), \tag{2.5}$$

where $X_j \subseteq \{x_1, x_2, \ldots, x_n\}$ is the set of all variables contained in the argument of local function f_j. Suppose that each local function of the factor graph is explicitly specified. The objective of the sum-product algorithm is then to compute the following *marginal* functions of g:

$$g_i(x_i) := \sum_{\{x_1, x_2, \ldots, x_n\} \setminus \{x_i\}} g(x_1, x_2, \ldots, x_n) \tag{2.6}$$

for every $i \in \{1, 2, \ldots, n\}$.

For example, when the sum-product algorithm is applied to the factor graph in Figure

2.2, the objective is to compute

$$
\begin{aligned}
g_1(x_1) &:= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4,x_5),\\
g_2(x_2) &:= \sum_{x_1}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4,x_5),\\
g_3(x_3) &:= \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4,x_5),\\
g_4(x_4) &:= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4,x_5) \text{ and}\\
g_5(x_5) &:= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_3,x_4,x_5).
\end{aligned}
$$

The sum-product algorithm is an iterative procedure operating on the factor graph representation of a global function, where the computation is performed by passing messages between every pair of connected function node and variable node. There are two types of messages, namely the message sent from a variable node v to a function node f — denoted by $\mu_{v\to f}$, and the message sent from a function node f to a variable node v — denoted by $\mu_{f\to v}$. We note that both message $\mu_{v\to f}$ and message $\mu_{f\to v}$ are functions of variable v.

We will use $\mathcal{N}(u)$ to denote the set of neighbors of a given node u in the factor graph. Using this notation, a message passed from a given node u to a given node w, where u and w are connected, will be computed only using the received messages from nodes $\mathcal{N}(u)\setminus\{w\}$. Specifically, the message passing rules are given as follows.

Function node update

$$\mu_{f\to v}(v) := \sum_{\mathcal{N}(f)\setminus\{v\}} f(\mathcal{N}(f)) \prod_{v'\in\mathcal{N}(f)\setminus\{v\}} \mu_{v'\to f}(v') \tag{2.7}$$

For example, in the portion of a factor graph, shown in Figure 2.3, the message passed

from function f to variable v is computed as

$$\mu_{f\to v}(v) = \sum_{u,u'u''} f(u,u',u'',v)\mu_{u\to f}(u)\mu_{u'\to f}(u')\mu_{u''\to f}(u'').$$

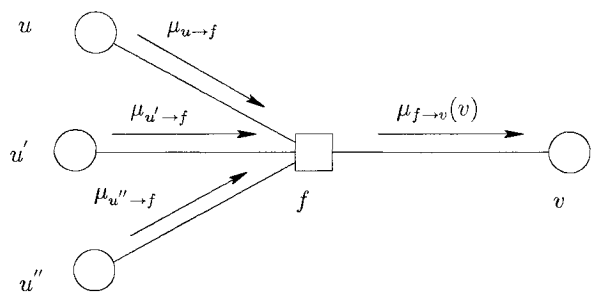

Figure 2.3: Function node f passes a message according to (2.7)

.

Variable node update

$$\mu_{v\to f}(v) := \prod_{f'\in\mathcal{N}(v)\setminus\{f\}} \mu_{f'\to v}(v) \tag{2.8}$$

For example, in the portion of a factor graph, shown in Figure 2.4, the message passed from function v to variable f is computed as

$$\mu_{v\to f}(v) = \mu_{h\to v}(v)\ \mu_{h'\to v}(v)\ \mu_{h''\to v}(v)$$

There is also the notion of *summary messages* that need to be computed during the execution of the sum-product algorithm. The summary message μ_v of a variable node v, again a function of variable v, is computed using the current incoming messages sent to node v, defined as follows.

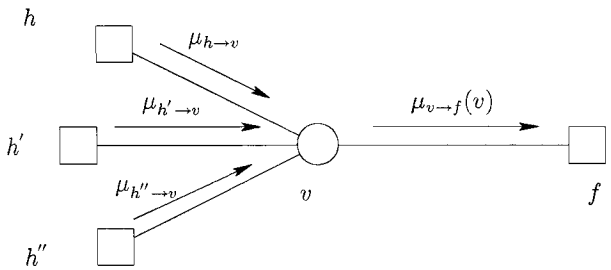

Figure 2.4: Variable node v passes a message according to (2.8)

Summary message

$$\mu_v(v) := \prod_{f\in\mathcal{N}(f)} \mu_{f\to v}(v). \tag{2.9}$$

For example, in the portion of a factor graph, shown in Figure 2.5, the summary message of variable v is computed as

$$\mu_v(v) = \mu_{h\to v}(v)\mu_{h'\to v}(v)\mu_{h''\to v}(v)\mu_{f\to v}(v).$$

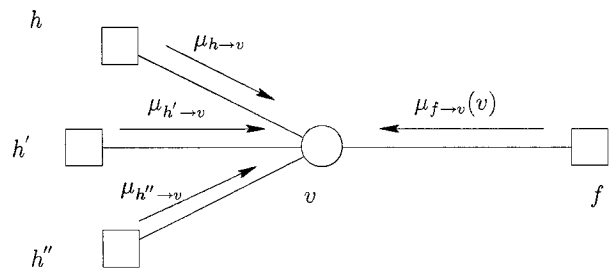

Figure 2.5: The computation of summary message μ_v for variable node v according to (2.9).

With these basic computation rules defined, the sum-product algorithm may organize these local message-passing rules in various ways, which often depends on the structure of the graph. Generically, any variant of the sum-product algorithm involves the following three phases:

- initialization phase, where a set of messages are initialized according to certain

choice,

- propagation phase, where messages are passed on the graph according to some order, and
- termination phase, where the passing of messages is terminated following some criterion.

If the factor graph is cycle-free, namely a tree, then the three phases are typically chosen as follows.

Sum-product algorithm on tree graphs

Initialization: Leaf nodes pass messages first. If a leaf node is variable node v, set $\mu_{v\to f}(v) := 1$, where f is the only neighbor of v; if a leaf node is a function node f, then set $\mu_{f\to v}(v) := f(v)$, where v is the only neighbor of f.

Propagation: A node only passes messages to a neighbor if the messages from all other neighbors have arrived.

Termination: When each variable node receives messages from all neighbors, it computes its summary messages and the algorithm terminates.

The following elegant result has been proved [8,14]: Let $\mathcal{G}$ be the factor graph representing the factorization of $g(x_1, x_2, \ldots, x_n)$ in (2.5), and suppose that $\mathcal{G}$ is a tree. Then when the sum-product algorithm described above is applied on $\mathcal{G}$, the resulting summary message $\mu_{x_i}(x_i)$ for each $x_i, i \in \{1, 2, \ldots, n\}$ is $g_i(x_i)$ in (2.6).

This result, governing the correctness of the sum-product algorithm on tree graphs, can be in fact understood quite intuitively. In essence, the sum-product algorithm is an efficient means of using the distributive law between multiplication and addition [15]. On one hand, multiplication is postponed to as late as possible, and on the other hand, the intermediate terms computed for calculating one g_i is efficiently saved and shared by the computation of other g_j's. For example, if one applies the sum-product algorithm on the factor graph in Figure 2.2, then it can be seen that for calculating g_1 and g_2, essentially

the following computations are carried out.

$$\begin{aligned} g_1(x_1) &= \sum_{x_2} f_1(x_1, x_2) \sum_{x_3} f_2(x_2, x_3) \sum_{x_4, x_5} f_3(x_3, x_4, x_5) \\ g_2(x_2) &= \sum_{x_1} f_1(x_1, x_2) \sum_{x_3} f_2(x_2, x_3) \sum_{x_4, x_5} f_3(x_3, x_4, x_5) \end{aligned}$$

It is worth noting that during the execution of the sum-product algorithm, the terms $\sum_{x_3} f_2(x_2, x_3)$ and $\sum_{x_4, x_5} f_3(x_3, x_4, x_5)$, commonly involved in computing both g_1 and g_2, are actually computed only once.

On factor graphs with cycles, initialization, propagation and termination phases may all be implemented with much richer families of variants [16–19], and we will postpone to later this chapter explaining one standard implementation of the sum-product algorithm for decoding codes represented by factor graphs with cycles. For factor graphs with cycles, we note that no known implementation of the sum-product algorithm results in the exact solutions for solving the marginals defined in (2.6). Nevertheless, experimental results have suggested that when the graph is large and sparse, the computation is approximately correct [8, 20]. In particular, if the objective is not to compute $g_i(x_i)$ for each i but to find the value of x_i that maximizes $g_i(x_i)$, the algorithm performs extremely well. This has been demonstrated by the decoding of codes on graphs [16].

2.2.3 Codes on graphs

Pioneered by the rediscovery of LDPC codes [16, 21, 22], the modern approach of coding and decoding has shifted to the paradigm of codes on graphs. Instead of presenting the complete historical development— featuring various special cases of factor graphs, we here choose to directly use the factor-graph language to describe this subject.

In this context, a widely used function in factor graphs is the *indicator function* $\delta[P]$, where for any boolean proposition P, $\delta[P]$ is evaluated to 1 if P is `true` and evaluated to 0 otherwise.

Given a linear block code C of length n and dimension k, there are two kinds of fundamental representations of C, namely, the generator matrix representations and the parity-check matrix representations. In fact both representations have a corresponding

factor graph. This will be explained using the following example, and the generalization beyond this special case is straight-forward.

Let the following matrices H and G be respectively a parity-check matrix and generator matrix of a binary linear block code C.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \tag{2.10}$$

and

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.11}$$

Based on H, each codeword $(c_1, c_2, \ldots, c_7)$ satisfies the following constraints.

$$c_1 \oplus c_2 \oplus c_3 = 0, \tag{2.12}$$

$$c_1 \oplus c_4 \oplus c_5 = 0, \tag{2.13}$$

$$c_1 \oplus c_6 \oplus c_7 = 0, \tag{2.14}$$

where $\oplus$ denotes addition modulo 2.

We may then translate these constraints to a set of indicator functions as follows.

$$f_1(c_1, c_2, c_3) := \delta[c_1 \oplus c_2 \oplus c_3 = 0], \tag{2.15}$$

$$f_2(c_1, c_4, c_5) := \delta[c_1 \oplus c_4 \oplus c_5 = 0], \tag{2.16}$$

$$f_3(c_1, c_6, c_7) := \delta[c_1 \oplus c_6 \oplus c_7 = 0]. \tag{2.17}$$

The product

$$f_1(c_1, c_2, c_3) f_2(c_1, c_4, c_5) f_3(c_1, c_6, c_7) \tag{2.18}$$

of these indicator functions then defines the code constraint of C. Thus this product can

be represented using a factor graph as shown in Figure 2.6.

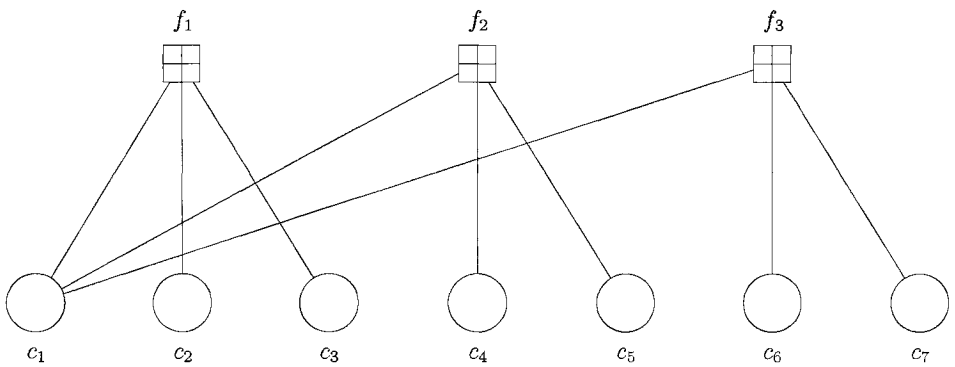

Figure 2.6: Factor graph corresponding to parity-check matrix H in (2.10). The "+" inside each function node is meant to indicate that the function corresponds to a check-sum constraint.

On the other hand, based on G, each codeword $(c_1, c_2, \ldots, c_7)$ satisfies the following constraints.

$$\begin{aligned} m_1 &= c_1, \\ m_1 \oplus m_3 &= c_2, \\ m_3 &= c_3, \\ m_1 \oplus m_2 &= c_4, \\ m_2 &= c_5, \\ m_4 &= c_6, \\ m_1 \oplus m_4 &= c_7, \end{aligned}$$

for any binary vector (m_1, m_2, m_3, m_4). — In fact, if the encoding is carried out using generator matrix G, then (m_1, m_2, m_3, m_4) is precisely the information vector to be transmitted.

We may then translate these constraints to a set of indicator functions as follows.

$$\begin{aligned} h_1(m_1, c_1) &:= \delta[m_1 \oplus c_1 = 0], \\ h_2(m_1, m_3, c_2) &:= \delta[m_1 \oplus m_3 \oplus c_2 = 0], \\ h_3(m_3, c_3) &:= \delta[m_3 \oplus c_3 = 0], \\ h_4(m_1, m_2, c_4) &:= \delta[m_1 \oplus m_2 \oplus c_4 = 0] \\ h_5(m_2, c_5) &:= \delta[m_2 \oplus c_5 = 0], \\ h_6(m_4, c_6) &:= \delta[m_4 \oplus c_6 = 0], \\ h_7(m_1, m_4, c_7) &:= \delta[m_1 \oplus m_4 \oplus c_7 = 0] \end{aligned}$$

The product

$$h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7) \tag{2.19}$$

of these indicator functions also defines the code constraint of C. Then this product can be represented using a factor graph as shown in Figure 2.7.

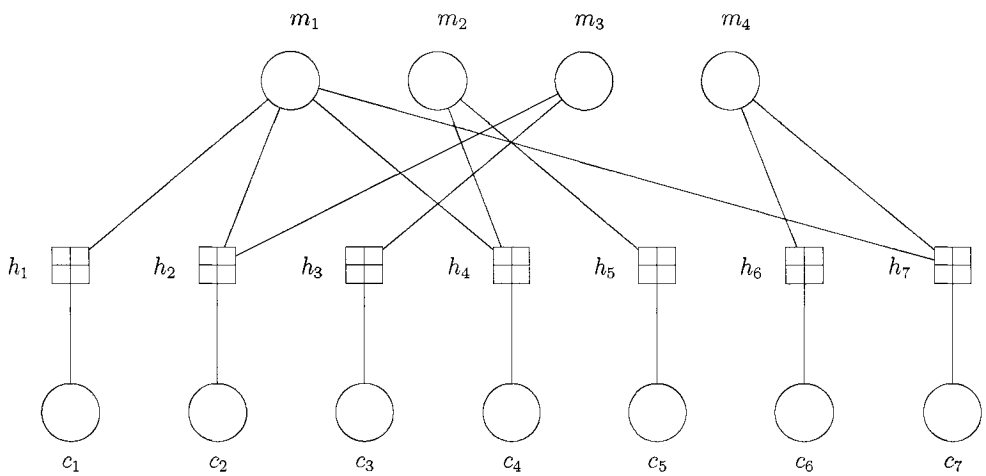

Figure 2.7: Factor graph corresponding to generator matrix G in (2.11)

In the context of codes on graph and particularly for randomly constructed codes such as LDPC codes and Fountain codes — both relevant to this work, the notion of

degree distribution is important. Specifically, the degree of a node u is defined as $|\mathcal{N}(u)|$, i.e., the number of neighbors of node u. On a factor graph representation of a code, we use λ_i to denote the fraction of edges connected to variable nodes of degree i, and ρ_j to denote the fraction of edges connected to function nodes of degree j. Clearly $\sum_i \lambda_i = 1$ and $\sum_j \rho_j = 1$, and hence the pair of sequences $\lambda_1, \lambda_2, \ldots$ and $\rho_1, \rho_2, \ldots$ is refer to as the *degree distribution* of the factor graph. Using the notion of degree distribution, the state-of-the-art codes — LDPC codes, can be defined.

Low-density parity-check (LDPC) codes

Low-density parity-check (LDPC) codes were originally introduced by Gallager [21], largely forgotten for thirty years, and then rediscovered recently [22]. In the modern perspective, LDPC codes are represented and decoded using factor graphs (or their equivalent). A family of LDPC codes is specified by a given degree distribution $(\{\lambda_i\}, \{\rho_j\})$ of factor graphs representing parity-check matrices. In LDPC literature, $\{\lambda_i\}$ is often referred to as the *left-degree distribution* and $\{\rho_j\}$ referred to as the *right-degree distribution*. It is easy to verify that given the left and right degree distributions, the rate of the family LDPC codes is a constant, independent of the length of the code (provided that all rows of the parity check matrices are linearly independent).

It has been reported that the asymptotic performance of LDPC codes can be characterized by the degree distribution of the codes, and there exist powerful methods [23] for predicting the performance of LDPC codes given a degree distribution $(\{\lambda_i\}, \{\rho_j\})$.

At long block lengths, LDPC codes are constructed at random such that the resulting factor graph had a desired degree distribution. At short block lengths, there are various algebraic and combinatorial techniques for constructing LDPC codes.

2.2.4 Iterative decoding of codes on graphs

The decoding of codes on graphs essentially uses the sum-product algorithm. This may be understand by formulating the decoding problem as the problem of simultaneously finding many marginals of a product function.

We will use the toy code C in the last subsection to illustrate this principle.

Suppose that a codeword $(c_1, c_2, \ldots c_7)$ is BPSK modulated to a symbol vector $(x_1, x_2, \ldots, x_7)$ and transmitted over a memoryless channel characterized by conditional probability distribution $p(y_i|x_i)$, where y_i is the received symbol for the transmitted symbol x_i. The objective of decoding may be formulated as obtaining an estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ of the transmitted codeword $(c_1, c_2, \ldots, c_7)$ and then declaring the transmitted (four) information bits. This communication diagram is shown in Figure 2.8. We note that BPSK modulation is a bijective function, which we will denote by ϕ. It is apparent that one may view the concatenation of BPSK modulation and channel $p(y_i|x_i)$ as a "virtual channel" characterized by the equivalent conditional distribution $p(y_i|c_i)$ given by

$$p(y_i|c_i) = \sum_{x_i \in \{1,-1\}} p(x_i|c_i)p(y_i|x_i),$$

where

$$p(x_i|c_i) = \delta[\phi(c_i) = x_i].$$

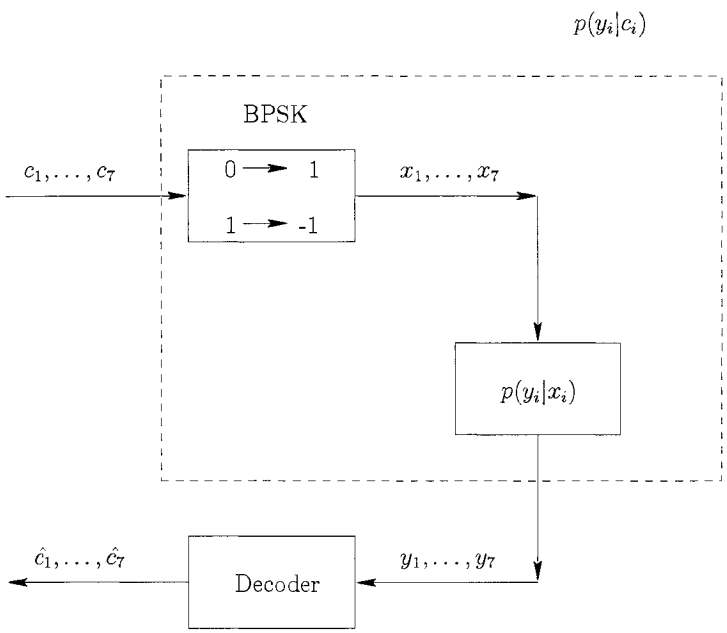

Figure 2.8: Example of communication diagram

Since obtaining the intended information bits from the estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ merely involves inverting the encoding function, we now only address how the estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ may be obtained.

Under the long-established maximum *a posteriori* probability (MAP) criterion [24], the estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ may be obtained as

$$\begin{aligned}
\hat{c}_1 &:= \arg\max_{c_1} p(c_1|y_1, \ldots, y_7) \\
\hat{c}_2 &:= \arg\max_{c_2} p(c_2|y_1, \ldots, y_7) \\
&\ldots \\
\hat{c}_7 &:= \arg\max_{c_7} p(c_7|y_1, \ldots, y_7)
\end{aligned}$$

Let us take the formulation of MAP estimate $\hat{c}_1$ and derive it further.

$$\begin{aligned}
\hat{c}_1 &= \arg\max_{c_1} p(c_1|y_1, \ldots, y_7) \\
&= \arg\max_{c_1} \frac{p(c_1, y_1, \ldots, y_7)}{p(y_1, \ldots, y_7)} \\
&= \arg\max_{c_1} P(c_1, y_1, \ldots, y_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(c_1, \ldots, c_7, y_1, \ldots, y_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(y_1, \ldots, y_7|c_1, \ldots, c_7) p(c_1, \ldots, c_7) \\
&\overset{\text{(memoryless channel)}}{=} \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(y_1|c_1) \ldots p(y_7|c_7) p(c_1, \ldots, c_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)
\end{aligned}$$

At this end, it is possible to relate the term $\sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$ above to a marginal of a function that can be represented by a factor graph.

Specifically, if we construct the factor graph of the code based on the parity-check matrix H in (2.10), then $p(c_1, c_2, \ldots, c_7)$ is equal to function $f_1(c_1, c_2, c_3)f_2(c_1, c_4, c_5)f_3(c_1, c_6, c_7)$ up to scale, under the usual assumption that each codeword is equally likely to be transmitted. Then clearly, the function $p(c_1, c_2, \ldots c_7)\prod_{i=1}^{7} p(y_i|c_i)$ is precisely the global function represented by the factor graph in Figure 2.9. Therefore computing

$$\sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7)\prod_{i=1}^{7} p(y_i|c_i)$$

is identical to computing a marginal of the global function represented by the factor graph. In the same way, obtaining the MAP estimate $\hat{c}_i$ for any other i can be reduced similarly to computing a corresponding marginal of the same global function. As such, these marginals can be computed simultaneously using the sum-product algorithm.

On the other hand, if we construct the factor graph of the code based on the generator matrix G in (2.11), then $p(c_1, c_2, \ldots, c_7)$ is equal to function

$$\sum_{m_1,m_2,m_3,m_4} h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7)$$

up to scale, again under the assumption that each codeword is equally likely. Denote product

$$h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7)$$

by $h(m_1, \ldots, m_4, c_1, \ldots, c_7)$. Then

$$\sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7)\prod_{i=1}^{7} p(y_i|c_i) = \sum_{c_2,\ldots,c_7,m_1,\ldots,m_4} h(m_1, \ldots, m_4, c_1, \ldots, c_7)\prod_{i=1}^{7} p(y_i|c_i).$$

In the above equation, the product

$$h(m_1, \ldots, m_4, c_1, \ldots, c_7)\prod_{i=1}^{7} p(y_i|c_i)$$

is the global function represented by the factor graph in Figure 2.10. Thus, computing the MAP estimate for c_1 and similarly for any other c_i reduces to determining a corresponding marginal of this global function, and the sum-product algorithm naturally applies.

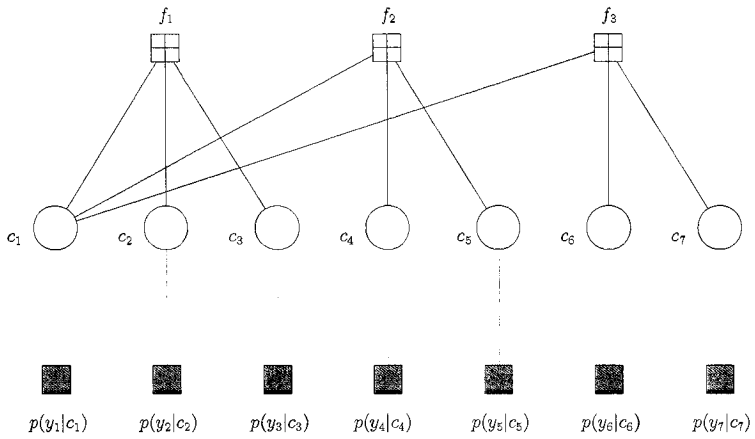

Figure 2.9: A factor graph representing $p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$ constructed from parity-check matrix H in (2.10).

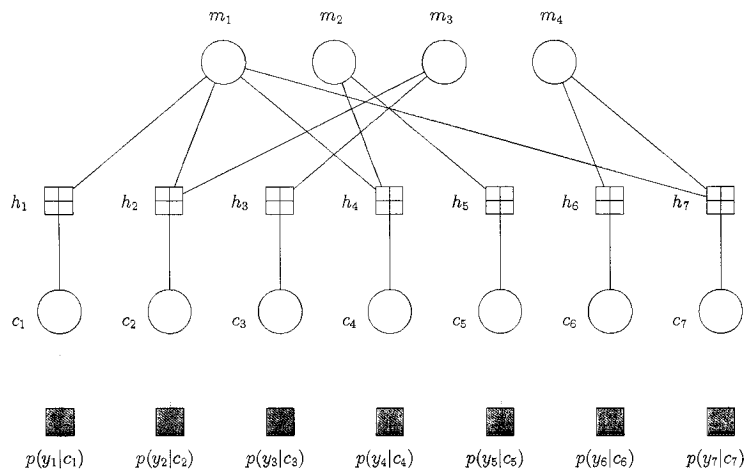

Figure 2.10: A factor graph representing $h(m_1, \ldots, m_4, c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$ constructed from generator matrix G in (2.11).

For codes defined on graphs, it is known that cycle-free graphs correspond to poor codes [25]. We now give a standard implementation of the sum-product algorithm for

decoding codes represented by graphs with cycles. For simplicity, we only focus on the case in which the factor graph is constructed from the parity-check matrix of the code.

Sum-product decoding on graphs with cycles (constructed from a parity-check matrix)

Initialization: Initialize all messages to constant 1 except those sent from the leaf function nodes representing function $p(y_i|c_i)$, which are set to the function $p(y_i|c_i)$ itself. These messages are often referred to as the *intrinsic information.*

Propagation: All variable nodes representing a codeword symbol send messages; then all function nodes representing a parity-check constraint send messages. Iterate this process.

Termination: When every message converge to a fixed point (up to scale) or when a pre-determined number of iterations is reached, compute the summary messages for all variable nodes, and terminate the algorithm.

Although the implementation above does not guarantee to result in the exact marginal functions desired for decoding, for codes represented by large and sparse graphs, experimental results have suggested that the performance is nearly optimal in the MAP sense. In fact, this implementation has been the standard approach for decoding the state-of-the-art error-correcting codes, the LDPC codes, and performance extremely close to Shannon's theoretical limit [26] has been demonstrated [16, 22, 27].

2.2.5 LT codes and Raptor codes over erasure channels

LT codes and Raptor codes are the existing two classes of Fountain codes, originally designed for erasure channels. Both being "rateless", these two class of codes differ in that a Raptor code is essentially an LT code with an LDPC precode.

LT codes

In a recent landmark paper [1], Luby introduced the first class of rateless codes for erasure channels, which he calls "LT codes". The encoding of LT codes, which Luby refers to as "LT process", produces an infinite stream of encoded symbols from k source symbols via `XOR` (modulo-2 addition) operations. In essence, an LT code may be characterized by a generator matrix G of k rows and infinite columns. To describe the encoding process, we will denote the binary source vector by $(b_1, b_2, \ldots, b_k)$ and the coded semi-infinite binary sequence by $c_1, c_2, \ldots$. The sub-matrix of G containing column 1 to column n will be denoted by G_n. Given G, the codeword $(c_1, \ldots, c_n)$ is generated by multiplying $(b_1, \ldots, b_k)$ with G_n. Thus the construction of an LT codes is equivalent to constructing G. We now describe how the i^{th} column of G can be generated for any i.

1. Choose randomly a positive number d from some distribution p_d.

2. Choose uniformly at random d distinct position in the i^{th} column and assign 1 to them. Assign 0 to the rest of the column.

To complete describing the construction of LT codes, we note that the distribution p_d used in generating matrix G is known as the Robust Soliton Distribution, invented by Luby in [1].

Apparently the encoding process up to codeword bit c_n defines a factor graph $\mathcal{G}_n$ involving variable nodes $c_1, \ldots c_n$ and $b_1, \ldots, b_k$. That is, $\mathcal{G}_n$ is the factor graph constructed from G_n. The decoding algorithm, at the n^{th} channel use, is precisely the sum-product algorithm applied to $\mathcal{G}_n$, and similar to the standard implementation for decoding LDPC codes. We note however that under erasure channel models, the channel law allows a significant reduced complexity comparing with standard LDPC decoding over Gaussian or binary symmetric channels. The reader is referred to [1] for the details of the reduced-complexity implementation. We here only remark that such an implementation in fact corresponds to that the decoding graph keeps growing with the received codeword bits, and at the same time, graph edges are progressively removed.

It has been shown that LT codes can achieve the capacity of any erasure channel without channel statistics known at the transmitter or receiver. However, the encoding

cost (measured in terms of number of XOR operations) per information bit grows logarithmically with the information block length k. This limits the application of LT codes in transmitting large files. A remedy of this limitation is a pre-coding scheme with LDPC codes, as has been utilized in Raptor codes.

Raptor codes

In Raptor codes [2], a k-bit information vector is first encoded by a high-rate LDPC code of length k', and the k'-bit vector is further encoded by an LT code with specifically designed p_d. The work of [2] shows that such a construction not only achieves the capacity of a erasure channel universally, but also has linear encoding cost, namely, that the average encoding cost per information bit is bounded by a constant. This behavior is due to that Raptor codes allow a certain fraction of errors uncorrectable by the LT code (thereby reducing encoding cost otherwise needed) and leaves those errors to be corrected by the LDPC code. The factor graph of an example Raptor code is shown in Figure 2.11.

In Raptor code literature, distribution p_d is represented as a polynomial

$$\Omega(x) := \sum_{d=1}^{\infty} \Omega_d x^d,$$

where Ω_d is $p_d(d)$. Table 2.1, taken from [2], list the optimized choices of $\Omega(x)$ for various information block length k.

The decoding of Raptor codes over erasure channels is nearly identical to that of LT codes, except that the graph that the algorithm operates on is one in the form of Figure 2.11.

At this end, we are ready to turn to the application of Raptor codes over fading channels, its limitation over such channels and our proposed solution.

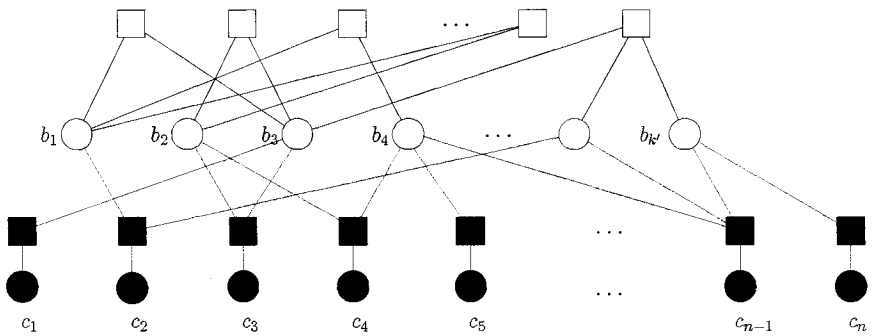

Figure 2.11: The factor graph of a Raptor code

k	65536	8000	100000	120000
Ω_1	0.007969	0.007544	0.006495	0.004807
Ω_2	0.493572	0.493610	0.495044	0.496472
Ω_3	0.166220	0.166458	0.168010	0.166912
Ω_4	0.072646	0.071243	0.067900	0.073374
Ω_5	0.082558	0.084913	0.089209	0.082206
Ω_8	0.056058	0.049633	0.041731	0.057471
Ω_9	0.037229	0.043365	0.050162	0.035951
Ω_{18}				0.001167
Ω_{19}	0.055590	0.045231	0.038837	0.054305
Ω_{20}		0.010157	0.015537	
Ω_{65}	0.025023			0.018235
Ω_{66}	0.003135	0.010479	0.016298	0.009100
Ω_{67}		0.017365	0.010777	

Table 2.1: $\Omega(x)$ of Raptor codes optimized for various values of k. Only the entries corresponding to $\Omega_d \neq 0$ are included in the table.

Chapter 3

Reduced-Complexity Decoding of Raptor Codes over Fading Channels

3.1 Existing algorithms

In Chapter 2.2.5, we introduced Raptor codes and decoding over erasure channels. On erasure channels, Raptor codes are known to be universally capacity-achieving. Raptor codes have recently been tested over binary symmetric channels, Gaussian channels and fading channels, and over all these channel models with excellent performances [7, 10, 11]. In particular, on Rayleigh fading channels. Raptor codes demonstrated a very good performance only within $10 - 15\%$ gap from averaged capacity at all SNR [7].

As noted in previous chapter, we now consider the scenario of point-to-point, single-antenna communication over fading channels, as described in (2.2), except that we will focus on only the slow fading case where h_i reduces to h.

Over such channels, we have commented that rateless codes can be an effective solution to resolve the trade-off between efficiency and reliability, as has also been demonstrated in the literature.

It is necessary to note that our setting is equivalent to the setting of AWGN channels with (receive) SNR $\gamma|h|^2$, known at the receiver but unknown at the transmitter. In addition, it has been shown that the realized rates of Raptor codes on Gaussian channels with any SNR is close to the Gaussian channel capacity and that the distribution of h,

except for inducing a distribution of realized rates, plays little role in the performance of Raptor codes [7]. As a consequence, our channel models essentially reduce to AWGN channels, parametrized by the (receive) SNR. Thus in what follows, we will simply treat our channel model as the equivalent *real* Gaussian channels.

For purpose of comparing the complexity of decoding, we chose a Raptor code as presented in [11]. Recall the Raptor codes diagram illustrated in Figure 2.11, the Raptor codes are LT codes combined with outer codes. Typically these outer codes are high-rate LDPC codes. Regarding the inner LT codes, we chose the distribution from Table 2.1. Here, we use the first configuration, which can be rewritten in Equation 3.1. For the outer pre-coding, we use a left regular degree distribution (variable nodes degree is 4 for all nodes) and right Poisson (function nodes randomly chosen with a uniform distribution).

$$\begin{aligned}\Omega(x) &= 0.007969x + 0.493572x^2 + 0.166220x^3 + 0.072646x^4 \\ &+ 0.082558x^5 + 0.056058x^8 + 0.037229x^9 \\ &+ 0.055590x^{19} + 0.025023x^{65} + 0.003135x^{66}\end{aligned} \tag{3.1}$$

The Raptor code has original $k = 9,500$ information bits $(a_1,a_2,\ldots,a_{9,500})$. Since the high-rate ($R = 0.95$) outer LDPC code, we add 500 redundant bits to get $k' = 10,000$ bits *intermediate* vector $(b_1,b_2,\ldots,b_{10,000})$ by pre-coding, and then, LT code has $10,000$ bits intermediate vector $(b_1,b_2,\ldots,b_{10,000})$ to encoded to infinite bits sequence $c_1,c_2,\ldots$ by multiplying with a sparse generator matrix, which has k' rows and may have infinite columns and the distribution of number of 1s in columns drawn from Equation 3.1.

In Chapter 2.2.4, we introduced the iterative decoding of codes on graphs. Equipped with the sum-product algorithm, easily, we can implement the decoding of Raptor codes over fading or Gaussian channels. For purpose of comparing with previous factor graphs, here, we fold the upper LDPC code down to the left side from Figure 2.11, thus, the factor graph becomes Figure 3.1.

The state-of-the-art decoding algorithm for Raptor codes over fading or Gaussian channels is, to the best of our knowledge, the standard belief propagation algorithm applied in every decoding attempt. A typical implementation of such an algorithm — which we will refer to Algorithm A throughout this thesis — is described next. We note

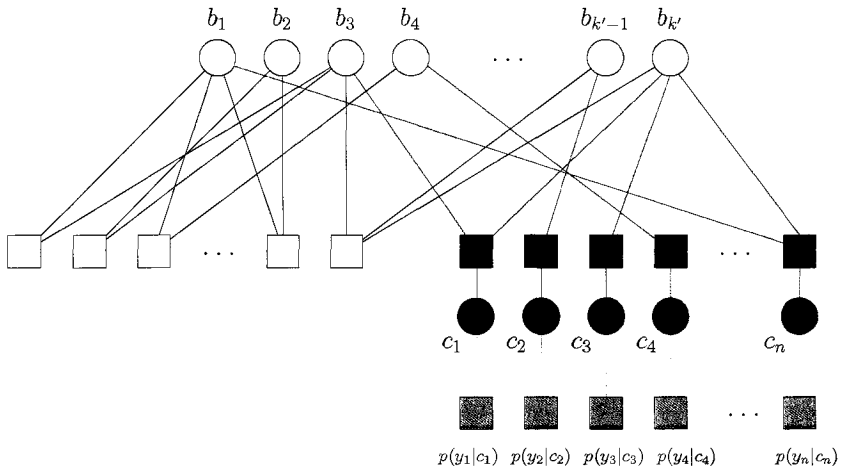

Figure 3.1: A detailed factor graph of a Raptor code. White circles represent LDPC codeword bits, white boxes represent the parity checks of the LDPC code, shaded circles represent the Raptor codeword bits, and shaded boxes represent LT code parity checks. The graph shows a Raptor code truncated to length n.

that hereafter we will always assume that BPSK modulation is used to transform c_i to x_i.

Algorithm A, which will also be referred to as Algorithm $A(T, L)$, is parametrized by a pair (T, L). Integer T is the interval between two consecutive decoding attempts. The time n^* for the first decoding attempt is chosen such that k/n^* is slightly lower than the realized capacity $C(h)$. That is, the l^{th} decoding attempt is at time $n_l := n^* + (l-1)T$. Integer L is the number of message-passing iterations performed in each decoding attempt, where an iteration is defined as all bit nodes passing messages to check nodes followed by all check nodes passing messages to bit nodes.

In the first iteration of *every* decoding attempt, the message passed from each bit node $b_i, i = 1 \ldots, k'$ along each of its edges is set to $(1, 1)$, and the message passed from each bit node $c_j, j = 1, \ldots, n_l$, is set to the intrinsic information (p_j, q_j) of c_j, where $p_j := \Pr[y_j|c_j = 0]$ and $q_j := \Pr[y_j|c_j = 1]$.

At the end of L iterations, the decoder computes the APP value for each bit b_i based on all messages passed to b_i and a hard decision $\hat{b}_i$ of b_i is then made. If vector $(\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{k'})$ fails to satisfy all parity checks in the factor graph, the decoder waits for the next decoding attempt to perform decoding again. If vector $(\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{k'})$ satisfies

all parity checks, then the decoder inverts the LDPC encoding function and obtains an estimate $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k)$ of the transmitted vector $(a_1, a_2, \ldots, a_k)$. We will assume that the decoder is able to determine whether the estimate $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k)$ is correct, which in practice corresponds to that a few CRC bits are embedded in $(a_1, a_2, \ldots, a_k)$.[1] If an error is detected in $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k)$, then the decoder again waits for the next decoding attempt to resume decoding. If no error is detected in $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k)$, the decoder declares $(\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k)$ as the transmitted message and signals the transmitter to terminate the transmission of this codeword.

Algorithm $A(T, L)$ can be briefly described by the following pseudo-code.

Algorithm A(T, L)

Initialization:
- Set all messages to (1, 1) on LDPC part
- Set all messages to (1, 1) on LT part

Propagation:
- **for** $i = 1$ **to** L
  - **do**
    - Function Nodes Update
      - on LT part
      - on LDPC part
    - Variable Nodes Update
      - on LT part
      - on LDPC part

Termination:
- Summarize all messages on LT variable nodes.
- **for** $i = 1$ **to** $10,000$
- **if** $p_i > q_i$, *for all* i
  - **then** Decoding successfully, and terminate the Algorithm
  - **else**
    - Receive T more bits
    - **go to** *Initialization*

It is possible that some variants of Algorithm A are considered in the literature (for example those stopping iterations based on some criterion of convergence), but these variants are all expected to be quite similar to what is described above. In particular, one expects little difference in performance or complexity, the metrics of which we will

[1]The rate loss due to such an implementation is neglected throughout this thesis.

make precise next.

3.2 Performance and complexity metrics

The performance of Raptor codes may be evaluated using the realized rate R and the word error rate as a function of rate R [7, 11]. More specifically, the word error rate at rate R is the probability that the transmitted message can not be decoded correctly provided that the transmission is forced to terminate and realize rate R.

We are interested in two different notions of decoding complexity, which we refer to as *decoding cost* and *decoding delay* respectively. The *decoding cost* is defined as the total number of iterations used for decoding a codeword over all decoding attempts. This notion of complexity approximately indicates the total amount of computation needed for decoding a codeword. Having various other impacts on hardware design, decoding cost also relates to the energy consumption of the decoder. The *decoding delay* is defined as the number of iterations in the final decoding attempt for decoding a codeword. This notion of complexity indicates the latency of completing decoding provided that the codeword can be decoded at this time. A high decoding delay will require a high clock speed to compensate.

3.3 Proposed algorithm and theoretical justification

Before presenting the modified decoding algorithm, we first examine the complexity of Algorithm A. The decoding delay of Algorithm A is clearly L. From existing results in literature, L is typically chosen to be around 100. In this work, we will investigate reducing L in Algorithm A so as to obtain a benchmark for complexity comparison with our proposed algorithm. As L keeps decreasing, one naturally expects a sacrifice of performance and as such, with Algorithm A, there is a trade-off between performance and decoding delay.

If a codeword is successfully decoded at the M^{th} decoding attempt, then the decoding cost using Algorithm A is ML. It is expected that decreasing T results in finer sampling

of realized rates, and the price to pay is an increased decoding cost, approximately proportional to $1/T$. Thus, fixing L in Algorithm A, there is a trade-off between performance and decoding cost. In existing results in the literature, T is typically chosen to be around 100. For the code parameter chosen in this paper, choosing T to be much smaller than 100 only results in negligible performance gain in practice. Thus in this paper, for any value of L, we will take $T = 100$ as the empirically optimal performance-cost trade-off point in Algorithm A, and make no attempt to modify it.

Now we present the modified algorithm, which we refer to as Algorithm B, aiming at reducing both decoding cost and decoding delay with no sacrifice of performance.

Algorithm B, which will also be referred to as Algorithm $B(T, L)$, is similarly parametrized by the pair (T, L), where T is again the time interval between two consecutive decoding attempts and L is the number of message-passing iterations at decoding attempt l, $l > 1$. In the first decoding attempt, Algorithm B is identical to Algorithm A, where the number of iterations is set to a relative large number (we use 100 in our implementation). At the l^{th} decoding attempt ($l > 1$), Algorithm B is *nearly* identical to Algorithm A with the only difference in initialization (first iteration) of passing messages. To be specific, we will use $\mathcal{G}_n$ to denote the factor graph of the Raptor code truncated at length n. In the first iteration of the l^{th} decoding attempt ($l > 1$), each bit node $c_j, j = 1, \ldots, n_l$, passes the intrinsic information in the same way as in Algorithm A. For each bit node $b_i, i = 1, \ldots, k'$, the message it passes to a neighbor check node v is the product of all incoming messages from check nodes in $\mathcal{N}_{l-1}(b_i) \setminus \{v\}$ in the final iteration of the $(l-1)^{\text{th}}$ decoding attempt, where $\mathcal{N}_{l-1}(b_i)$ is the set of all neighbors of b_i in factor graph $\mathcal{G}_{n_{l-1}}$. In simple terms, unlike Algorithm A which resets decoder at each decoding attempt, Algorithm B simply "continues" decoding from the results of last decoding attempt.

Algorithm $B(T, L)$ can be briefly described by the following pseudo-code.

Algorithm B(T, L)

comment: Set Iteration number=100

Initialization:
- Set all messages to (1,1) on LDPC part
- Set all messages to (1, 1) on LT part

Propagation:
- **for** $i = 1$ **to** *Iteration number*
- **do**
  - Function Nodes Update
    - on LT part
    - on LDPC part
  - Variable Nodes Update
    - on LT part
    - on LDPC part

Termination:
- Summarize all messages on LT variable nodes.
- **for** $i = 1$ **to** $10,000$
- **if** $p_i > q_i$, *for all* i
- **then** Decoding successfully, and terminate the Algorithm
- **else**
  - Receive T more bits, reset Iteration number=L
  - **go to** *Propagation*

We now provide some justification for Algorithm B.

Let D_n denote the diameter of factor graph $\mathcal{G}_n$, namely, the maximal number of edges along the shortest path between any two nodes on graph $\mathcal{G}_n$. Denote by $\epsilon_i^A(n; T, L)$ (and resp. by $\epsilon_i^B(n; T, L)$) the probability of decoding error for bit b_i if a hard-decision is made for b_i at decoding time n using Algorithm $A(T, L)$ (resp. Algorithm $B(T, L)$). Then we have the following results, the proof of which is presented in the final section of this thesis.

Theorem 1 *Suppose that both Algorithm* $A(T_A, L_A)$ *and Algorithm* $B(T_B, L_B)$ *attempt decoding at some time* n*, and that Algorithm* $B(T_B, L_B)$ *also attempts decoding at some later time* $n' > n$*. If* $\mathcal{G}_{n'}$ *is cycle-free and*

$$n' - n \geq (T_B/L_B) \min(D_n/2, L_A),$$

then $\epsilon_i^B(n'; T_B, L_B) \leq \epsilon_i^A(n; T_A, L_A)$ *for all* i *and all* T_A.

Although the theorem is based on the unrealistic assumption that the factor graph is cycle-free, one in fact expects it to hold approximately true even for graphs with cycles, provided the graph is sparse — as is standard, randomly constructed sparse graphs may be viewed as being locally cycle-free, and such assumptions underlie most analysis techniques for graphical codes. For a large sparse graph with cycles, as an approximation, one may also replace D_n in the theorem with a notion of "effective diameter" $\tilde{D}_n$ of graph $\mathcal{G}_n$, which approximates the maximal "effective distance" that a message propagates. It is expected that such an "effective diameter" of a sparse graph is much smaller than the true diameter of the graph. Roughly we may estimate $\tilde{D}_n$ by equating $\tilde{D}_n/2$ with the typical maximal number of iterations for message passing to converge (under Algorithm A) or to give consistent error performance.

This theorem suggests that if we we give Algorithm B some extra time, then it guarantees to give lower probability of error for each bit. We note however that the bound in the theorem is in essence a worst-case analysis and in fact quite loose. As we will show in our simulations, it is often possible for Algorithm B to outperform Algorithm A without the need of the extra time. Nevertheless, some insights may be obtained from the theorem, as we will now elaborate.

In the theorem, it is remarkable that the lower bound of extra time needed in Algorithm B only depends on the ratio T_B/L_B and given this ratio, the bound is independent of T_B or L_B.[2] In fact, this ratio — rather than T_B or L_B alone — plays the central role in the performance-complexity trade-off of Algorithm B. Specifically, the decoding cost of Algorithm B at time n' in the theorem is $(n' - n^*) \times (L_B/T_B)$, ignoring the constant number of iterations in the first decoding attempt. Thus, the decoding cost for Algorithm $B(T_B, L_B)$ to successfully decode at time n' is inversely proportional to the ratio T_B/L_B, and one would like to increase T_B/L_B ratio to reduce decoding cost. But on the other hand, as T_B/L_B increases, the bound in the theorem (if tight) indicates that a larger amount of extra time is needed for Algorithm B to guarantee a better bit error rate than Algorithm A; and this amount of extra time immediately translates to a loss of realized rates. The role of ratio T_B/L_B in performance-complexity trade-off is then evident.

[2]except that $n' - n$ needs to be a multiple of T_B.

Theorem 2 *Suppose that Algorithm A and Algorithm B are both parametrized by (T, L), and that the number of iterations in the first decoding attempt of Algorithm B is set to no smaller than L. At any decoding time n, if $\mathcal{G}_n$ is cycle-free, then $\epsilon_i^B(n;T,L) \leq \epsilon_i^A(n;T,L)$ for all i.*

The proof of this theorem is also postponed to the end of this chapter. The theorem suggests that on cycle-free factor graphs, when choosing the same parameter setting for the two algorithms, Algorithm B performs no worse than Algorithm A at all decoding times.

3.4 Proofs

To prove the theorems, we first establish some elementary results.

Let X, Y, and Z be three random variables where X takes values from $\{0,1\}$, and Y and Z take values from Ω_Y and Ω_Z respectively. Let function $M_Y : \Omega_Y \rightarrow \{0,1\}$ be an MAP estimate of X based on an observed value $y \in \Omega_Y$, and let function $\Omega_{YZ} : \Omega_Y \times \Omega_Z \rightarrow \{0,1\}$ be an MAP estimate of X based on a pair $(y,z) \in \Omega_Y \times \Omega_Z$. Clearly that functions M_Y and M_{YZ} depend on the joint distribution p_{XYZ}. Denote by E_X and E_{YZ} be respectively the error probability using estimator M_Y and estimator M_{YZ}.

Lemma 1 $E_{YZ} \leq E_Y$.

This result is a simple consequence of the MAP principle, and can be proved as below.
Proof: First we note the well-known fact that an MAP estimator by definition minimizes the probability of estimation error. Then by noticing that M_Y belongs to the set of all estimators of X based on observed pair $(y,z) \in \Omega_Y \times \Omega_Z$ (in which observation z is simply ignored), we have $E_{YZ} \leq E_Y$. □

This result simply suggests that when conditioned on extra evidence, MAP estimate can be more reliable. We now prove Theorem 1 using this result.
Proof: Graph $\mathcal{G}_{n'}$ being cycle-free implies that $\mathcal{G}_n$ is also cycle-free. It is well-known that using Algorithm A, message passing on $\mathcal{G}_n$ converges within $D_n/2$ iterations under Algorithm A if $L_A > D_n/2$, and all messages arriving at every node b_i give rise to (up

to scale) the conditional probability mass function (PMF) of b_i given all observed values $y_j, j \in \{1, \ldots, n\}$ [8]. On the other hand, if using Algorithm A with $L_A < n/2$ on $\mathcal{G}_n$, then it can be shown that message passing on $\mathcal{G}_n$ for L_A iterations gives rise to the conditional PMF of each b_i given (in general) a subset of all values $y_j, j \in \{1, \ldots, n\}$. In either of the two cases, we will denote by U_i the set of variables that are conditioned upon in the resulting conditional PMF for b_i. Thus, the hard decision on b_i after L_A iteration of Algorithm A on $\mathcal{G}_n$ is an MAP estimate of b_i given the values of U_i.

Now if the condition $n' - n \geq (T_B/L_B) \min(D_n/2, L_A)$ is satisfied and when Algorithm B is applied, the summary message at variable b_i is the conditional PMF of b_i given some other set U_i' of y_j variables. It can be verified that U_i is a subset of U_i'. This relationship between U_i and U_i' is sketched in an "unwrapped graph" in Figure 3.2. Thus, the hard decision on b_i is an MAP estimate of b_i given the values of U_i. By the above lemma, we see that the MAP estimate resulted from Algorithm B is no worse than MAP estimate resulted from Algorithm A, in terms of probability of error. □

The proof of Theorem 2 contains the same key ingredient as that of Theorem 1.

Proof:

We note that in the setting of the theorem, when using Algorithm A, the summary message at a variable b_i is resulted from precisely the set of the observations (y_j's) with distance less than $2L$ away. Denote these y_j's by U_i. Thus the summary message at variable b_i is precisely the conditional PMF of b_i given the values of U_i, and the hard decision on b_i is precisely the MAP estimate of b_i given the values of the U_i.

On the other hand, when using Algorithm B, some messages from observations (y_j's) farther than distance $2L$ away also have propagated to variable b_i. Thus the resulting summary message at variable b_i is the conditional PMF of b_i given the observed values of some other set U_i', which strictly contains U_i. The hard decision on b_i is the MAP estimate of b_i given U_i'.

By Lemma 1, we see that conditioned on U_i', the MAP estimate can not be worse than that conditioned on U_i. □

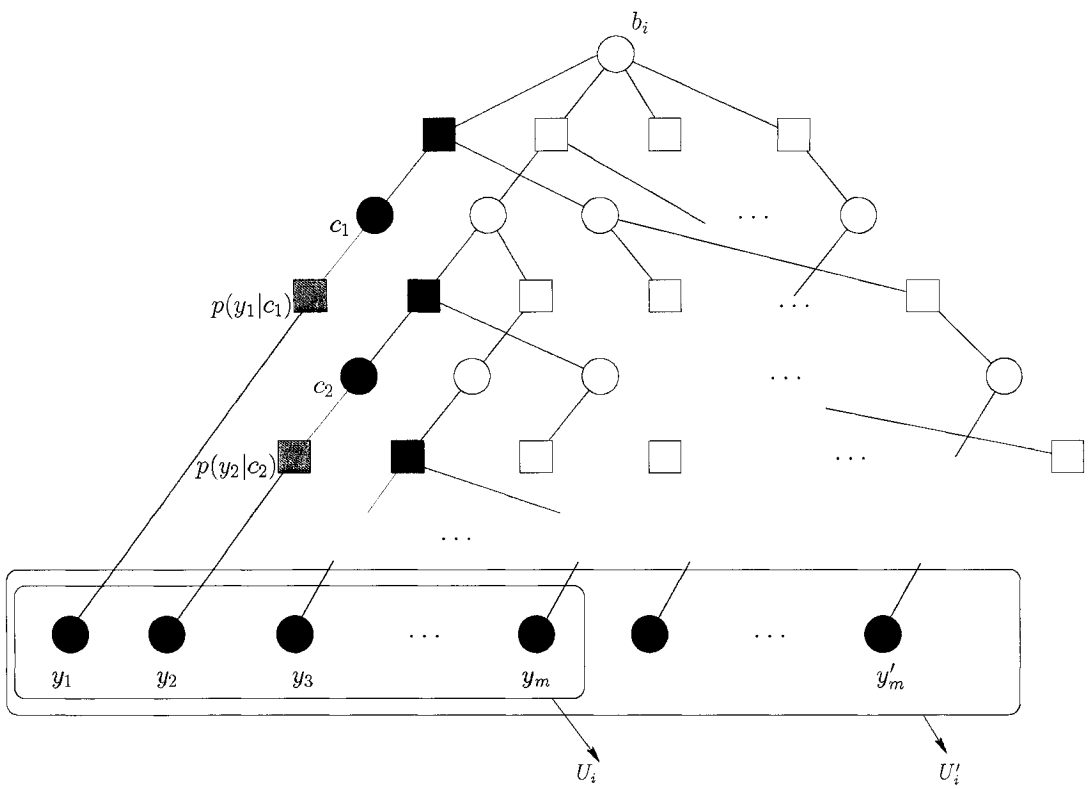

Figure 3.2: A portion of the unwrapped factor graph of a Raptor code, depicting the relationship between U_i and U'_i in the proof of Theorem 1.

Chapter 4

Simulation Results and Discussion

4.1 Simulation Setup

We performed Monte Carlo simulations for both Algorithm A and Algorithm B with various parameter settings and at receive SNR values corresponding to capacity 0.25, 0.5, and 0.75 bits/channel use. For a fair comparison, at each SNR value and for each simulated codeword transmission, all decoders (each using one of the two algorithms and at some parameter setting) perform decoding simultaneously on the channel output stream. That is, when any two decoders perform a decoding attempt of a codeword at the same time instant, the same channel output sequence is processed by the two decoders. At each SNR, the time n^* of the first decoding attempt is set to be identical across all decoders. Over channels with capacity 0.5, and 0.75 bits/channel use, we tested over 200 codeword. On channel with capacity 0.25 bits/channel use, we only tested over 120 codewords since the time consuming.

4.2 Results

Simulation results for channel with capacity 0.5 bits/channel use are presented in Figures (4.2) to (4.9). Also, Figures (4.10) to (4.14) present the simulation results for channel with capacity 0.75 bits/channel use. Last, the simulation results for channel with capacity 0.25 bits/channel use are presented in Figures (4.15) to (4.20).

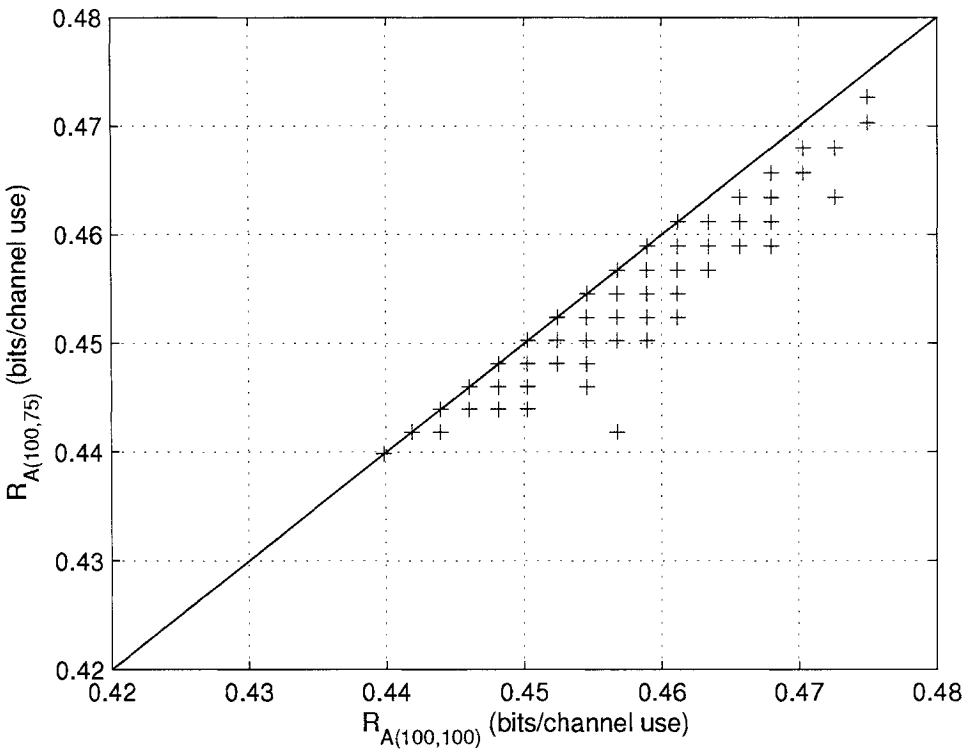

Figure 4.1: Scatter plot of realized rate by Algorithm $A(100, 75)$ vs realized rate by Algorithm $A(100, 100)$ over channel with capacity 0.5 bits/channel use.

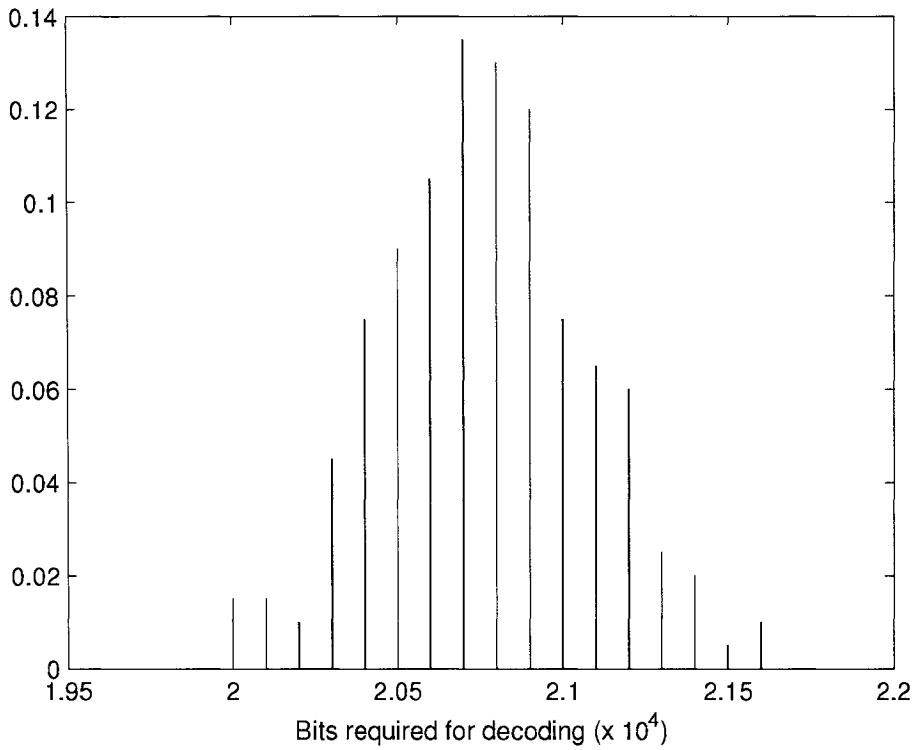

Figure 4.2: Histogram of number of bits required for successful decoding of Raptor code with $k = 9500$ on an AWGN with $E_s/N_0 = -2.83dB$ (capacity is 0.5 bits/channel use).

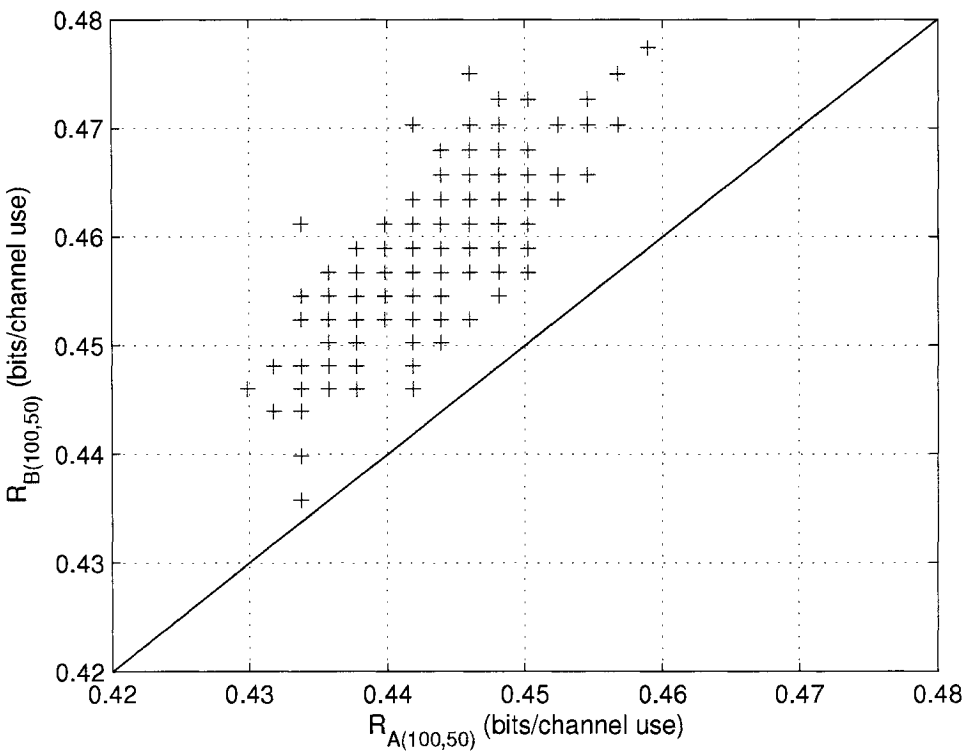

Figure 4.3: Scatter plot of realized rate by Algorithm $B(100, 50)$ vs realized rate by Algorithm $A(100, 50)$ over channel with capacity 0.5 bits/channel use.

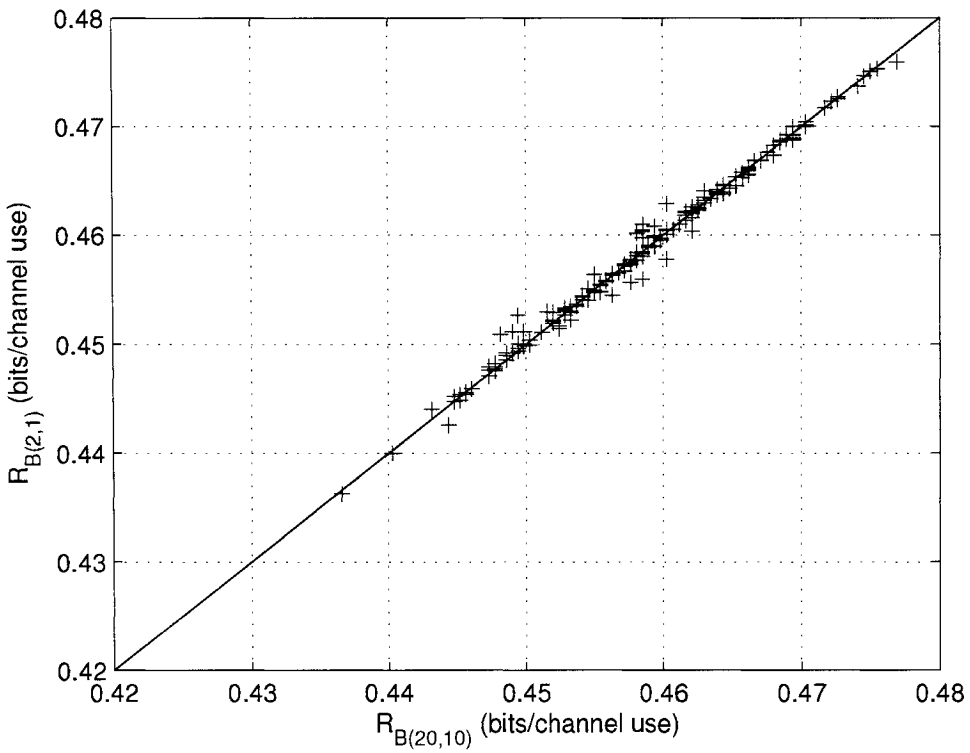

Figure 4.4: Scatter plot of realized rate by Algorithm $B(2, 1)$ vs realized rate by Algorithm $B(20, 10)$ over channel with capacity 0.5 bits/channel use.

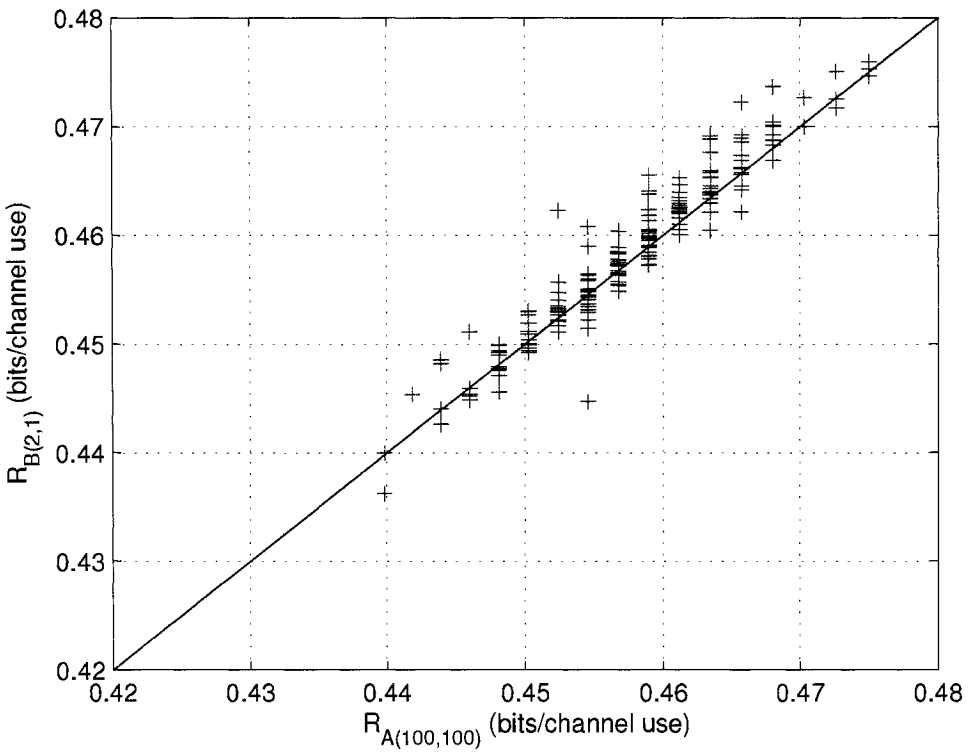

Figure 4.5: Scatter plot of realized rate by Algorithm $B(2, 1)$ vs realized rate by Algorithm $A(100, 100)$ over channel with capacity 0.5 bits/channel use.

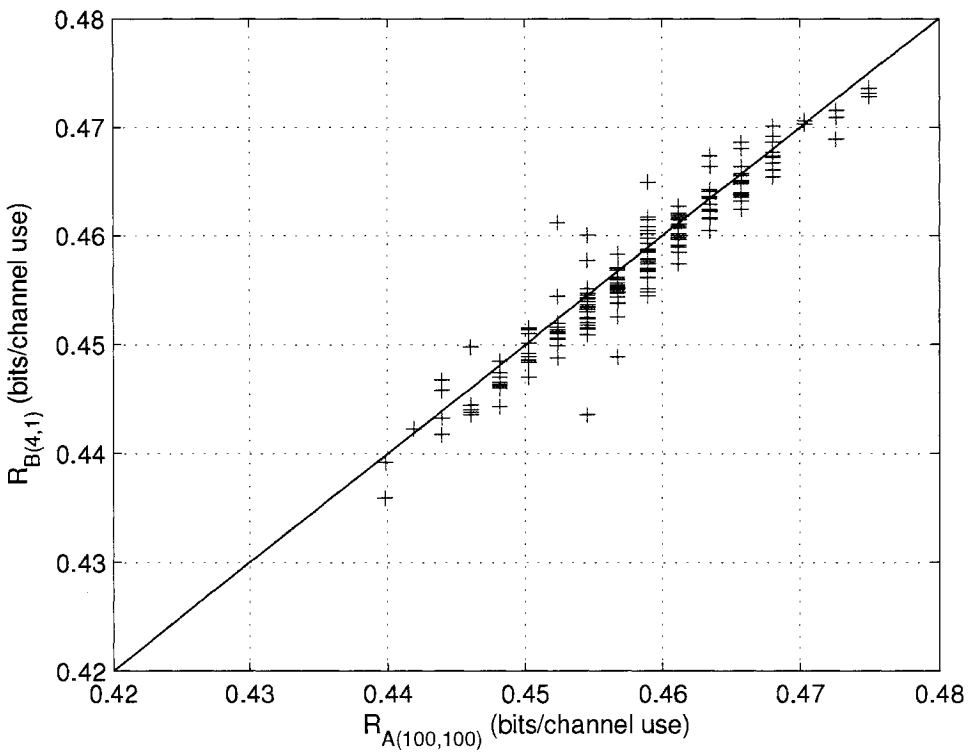

Figure 4.6: Scatter plot of realized rate by Algorithm $B(4,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.5 bits/channel use.

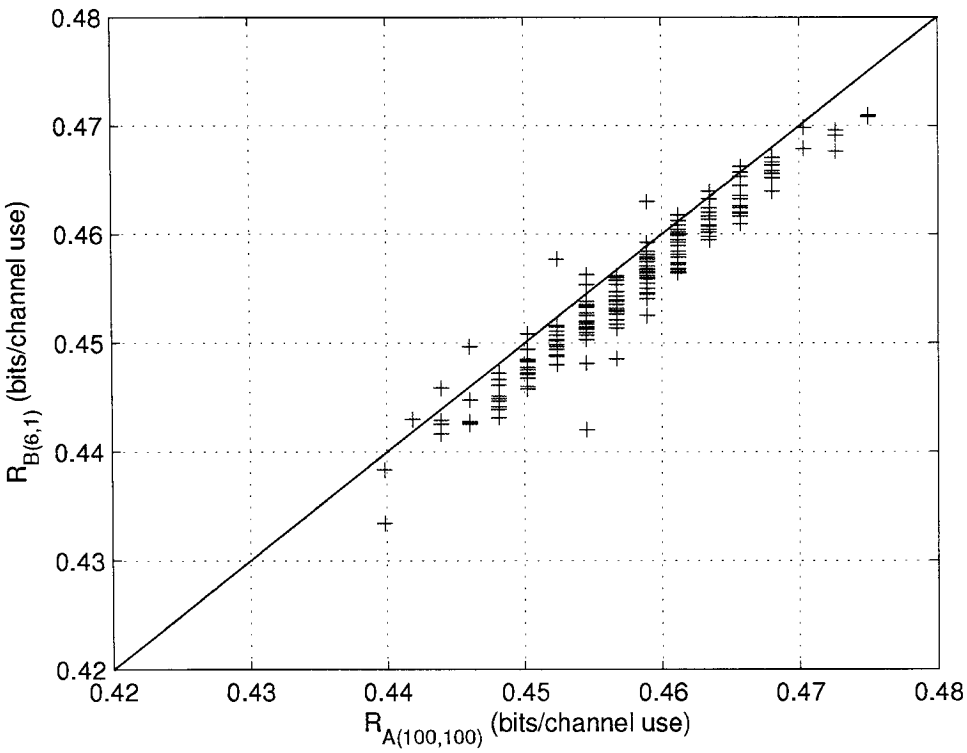

Figure 4.7: Scatter plot of realized rate by Algorithm $B(6,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.5 bits/channel use.

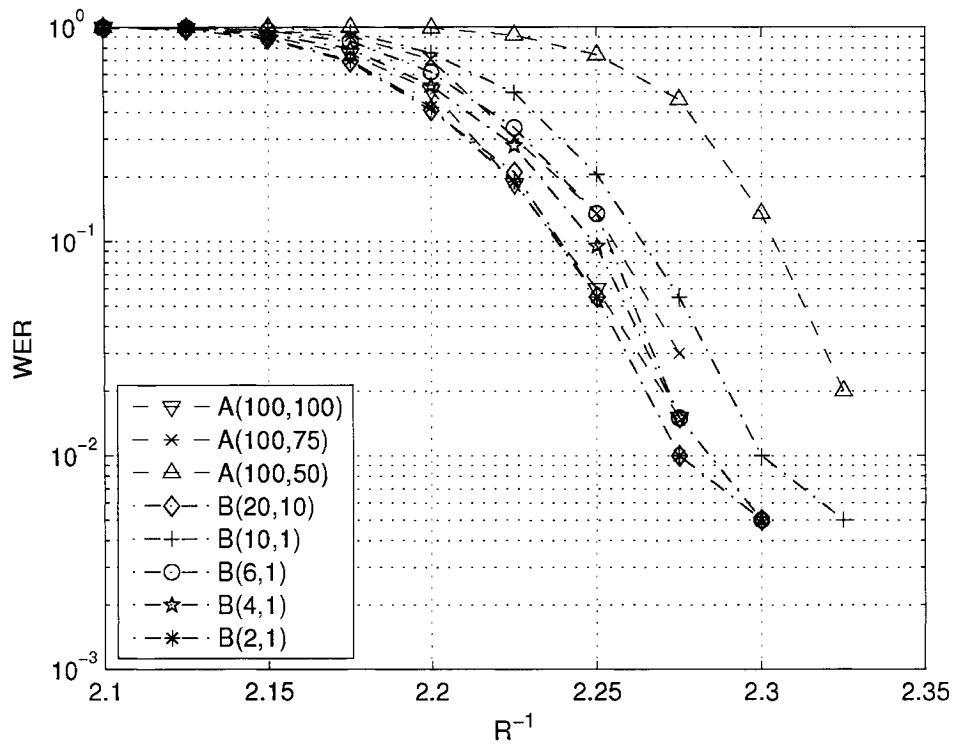

Figure 4.8: Word error rate as a function of $1/R$ for the two algorithms over channel with capacity 0.5 bits/channel use.

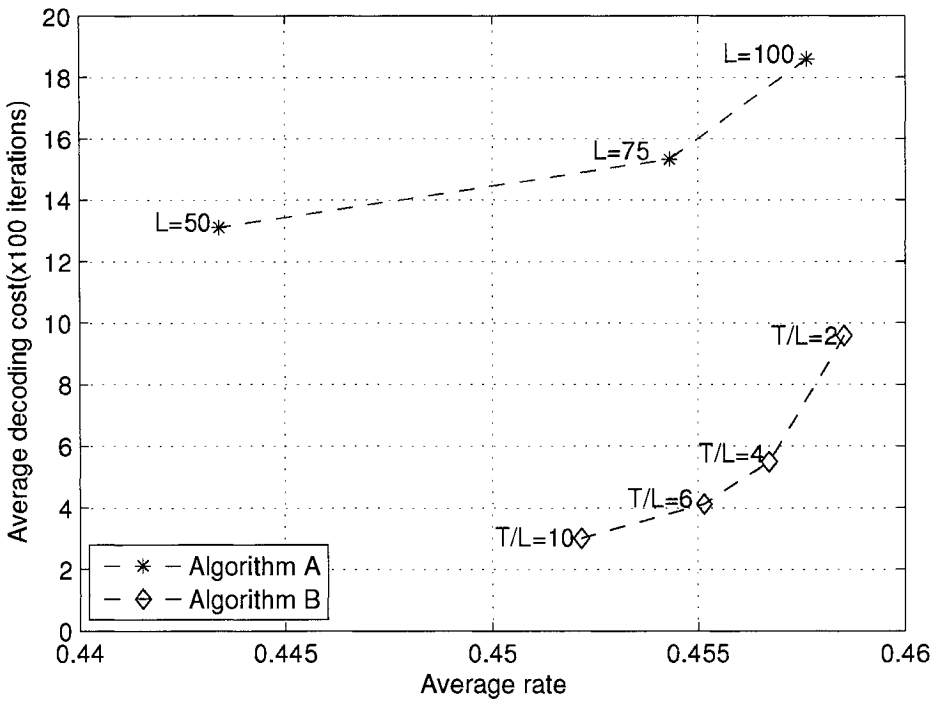

Figure 4.9: Average decoding cost vs average realized rates for the two algorithms over channel with capacity 0.5 bits/channel use.

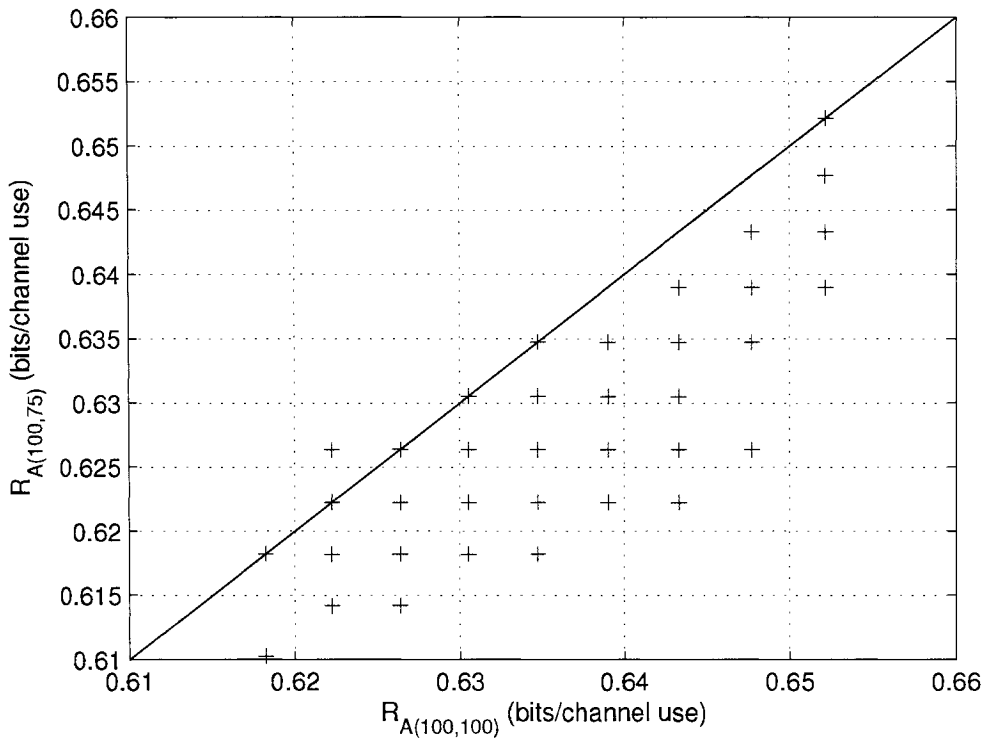

Figure 4.10: Scatter plot of realized rate by Algorithm $A(100, 75)$ vs realized rate by Algorithm $A(100, 100)$ over channel with capacity 0.75 bits/channel use.

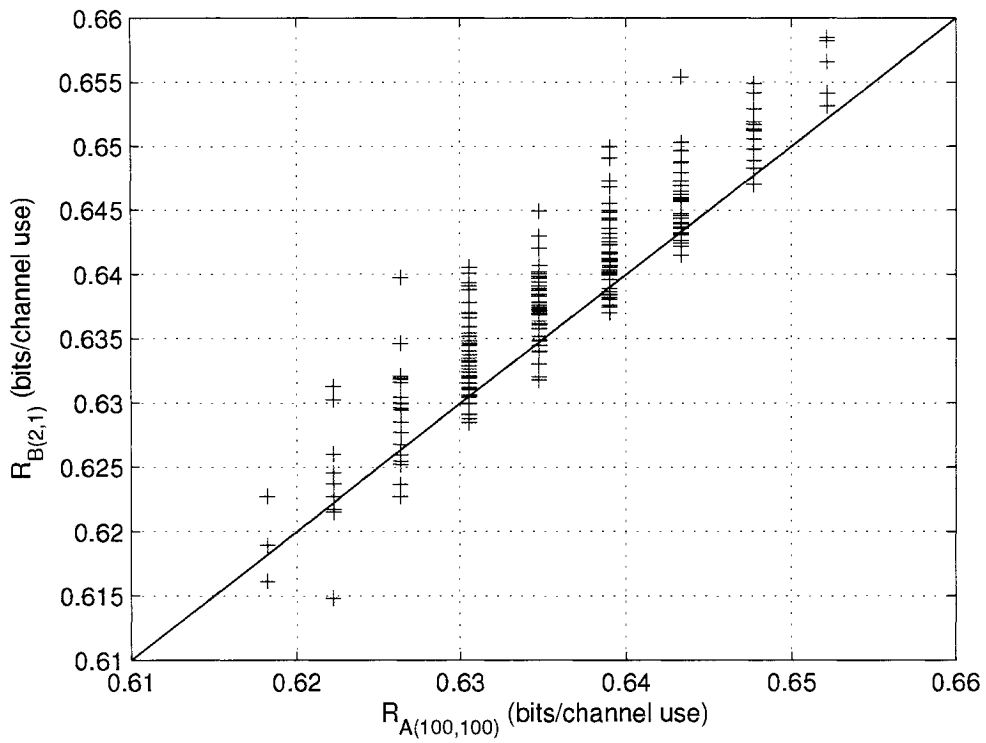

Figure 4.11: Scatter plot of realized rate by Algorithm $B(2,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.75 bits/channel use.

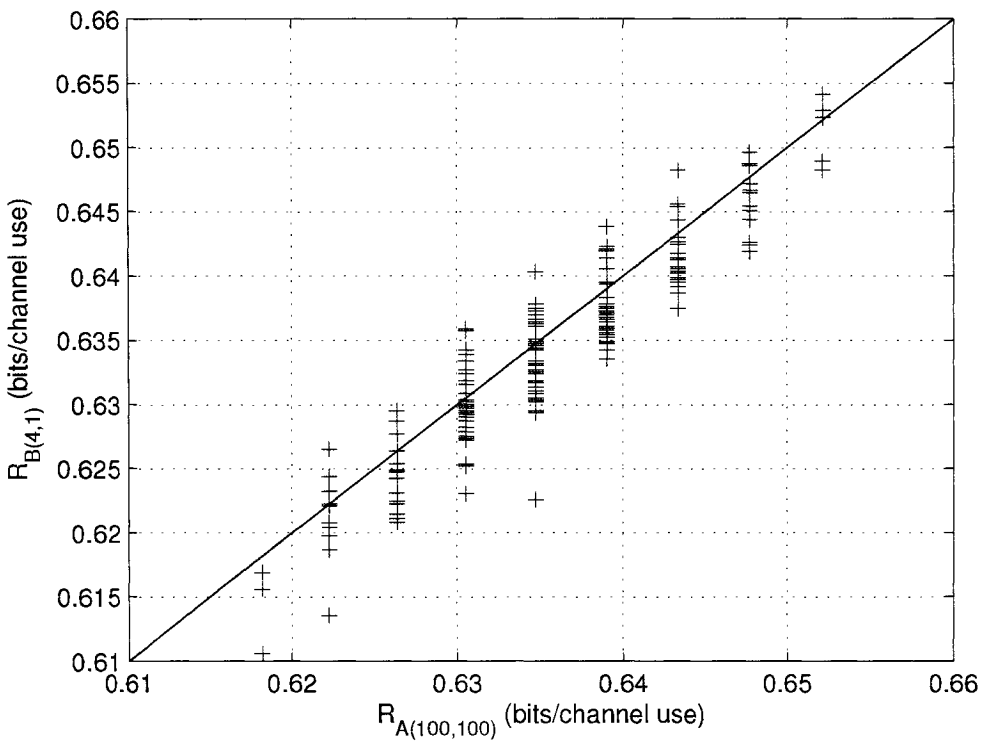

Figure 4.12: Scatter plot of realized rate by Algorithm $B(4,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.75 bits/channel use.

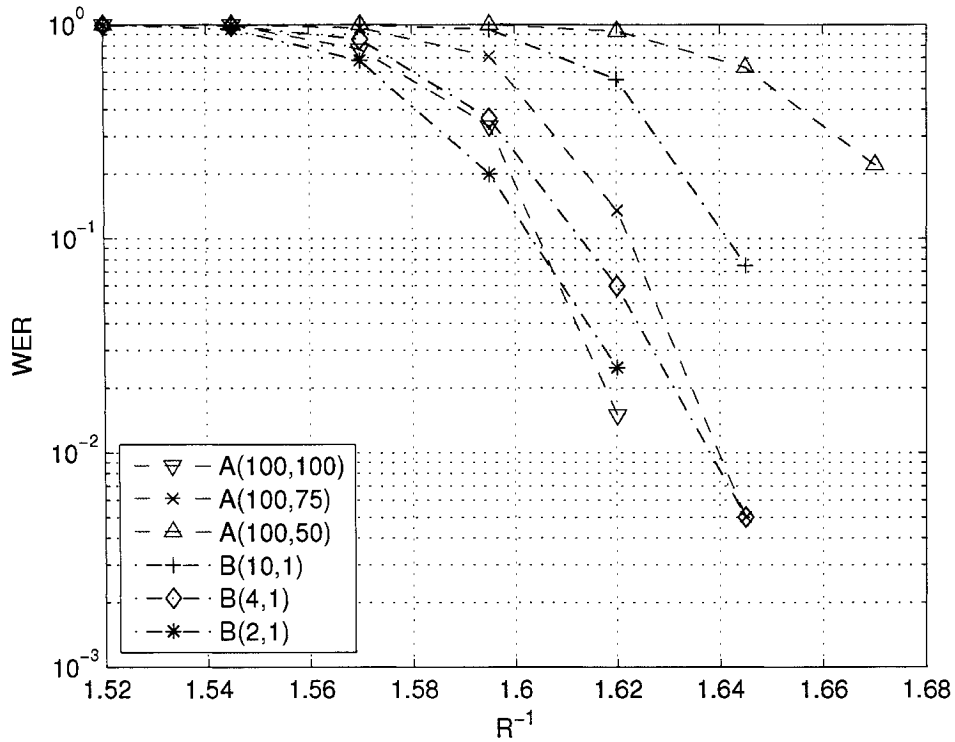

Figure 4.13: Word error rate as a function of $1/R$ for the two algorithms over channel with capacity 0.75 bits/channel use.

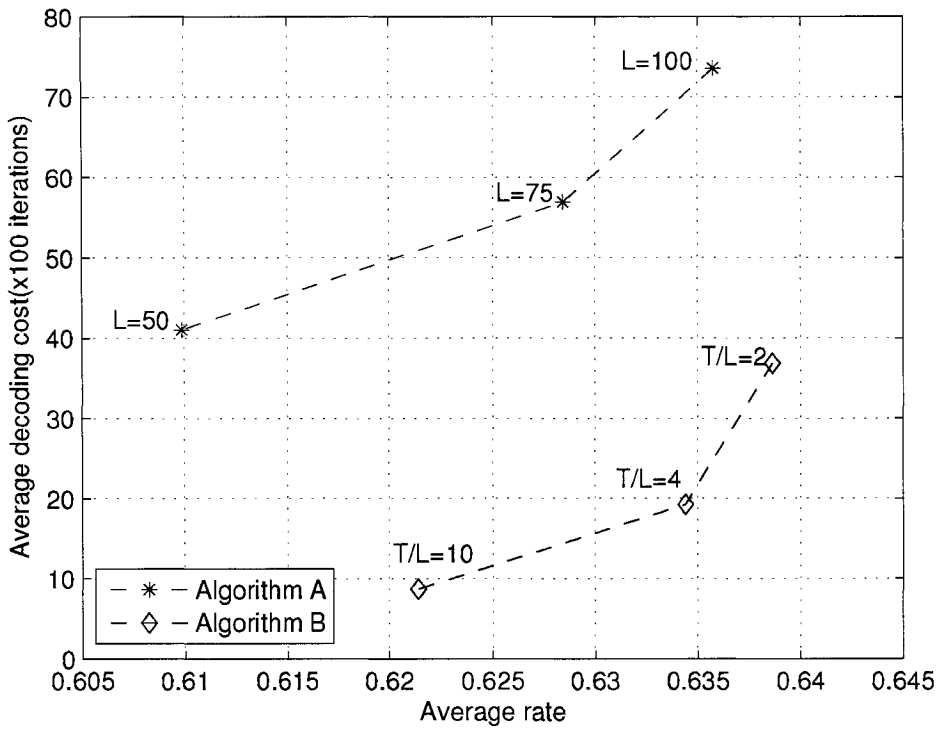

Figure 4.14: Average decoding cost vs average realized rates for the two algorithms over channel with capacity 0.75 bits/channel use.

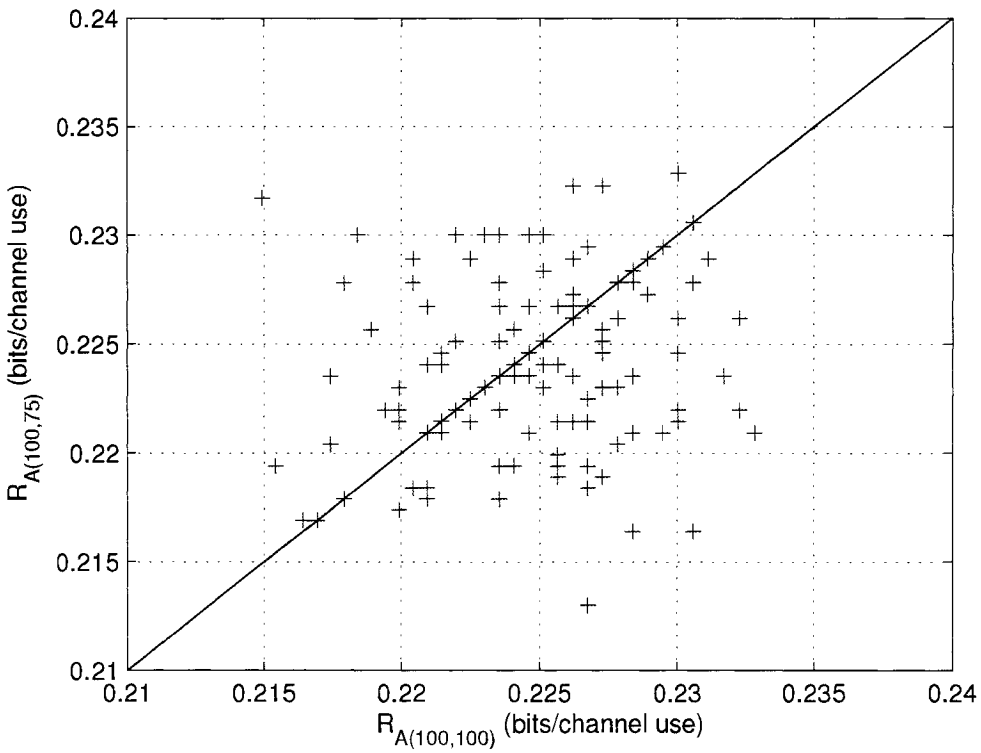

Figure 4.15: Scatter plot of realized rate by Algorithm $A(100, 75)$ vs realized rate by Algorithm $A(100, 100)$ over channel with capacity 0.25 bits/channel use.

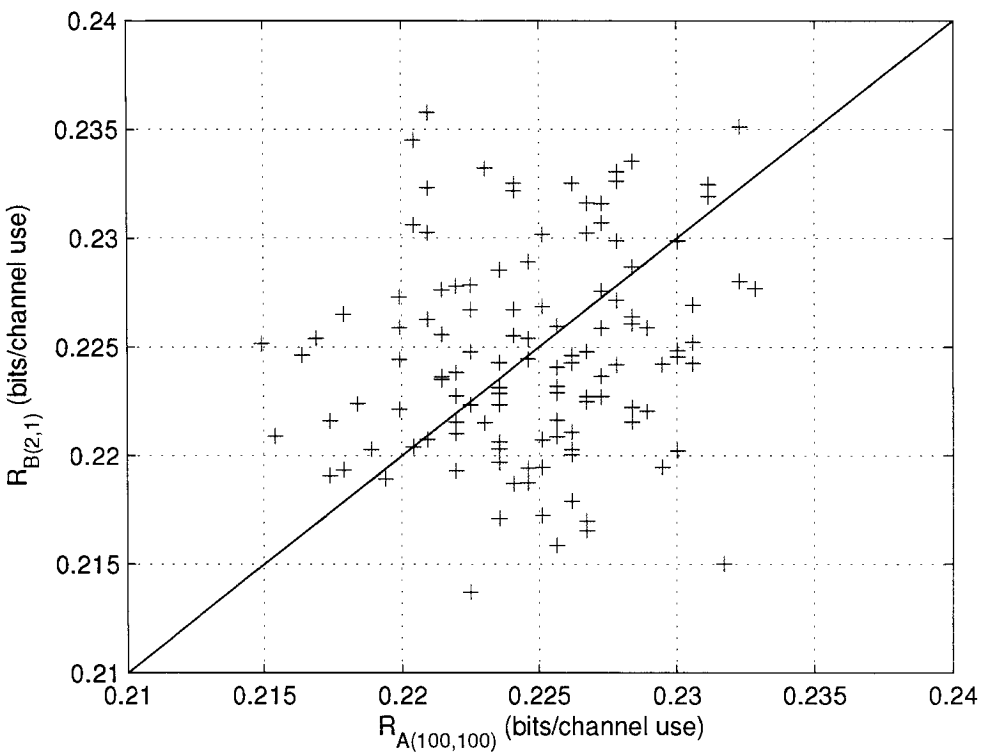

Figure 4.16: Scatter plot of realized rate by Algorithm $B(2,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.25 bits/channel use.

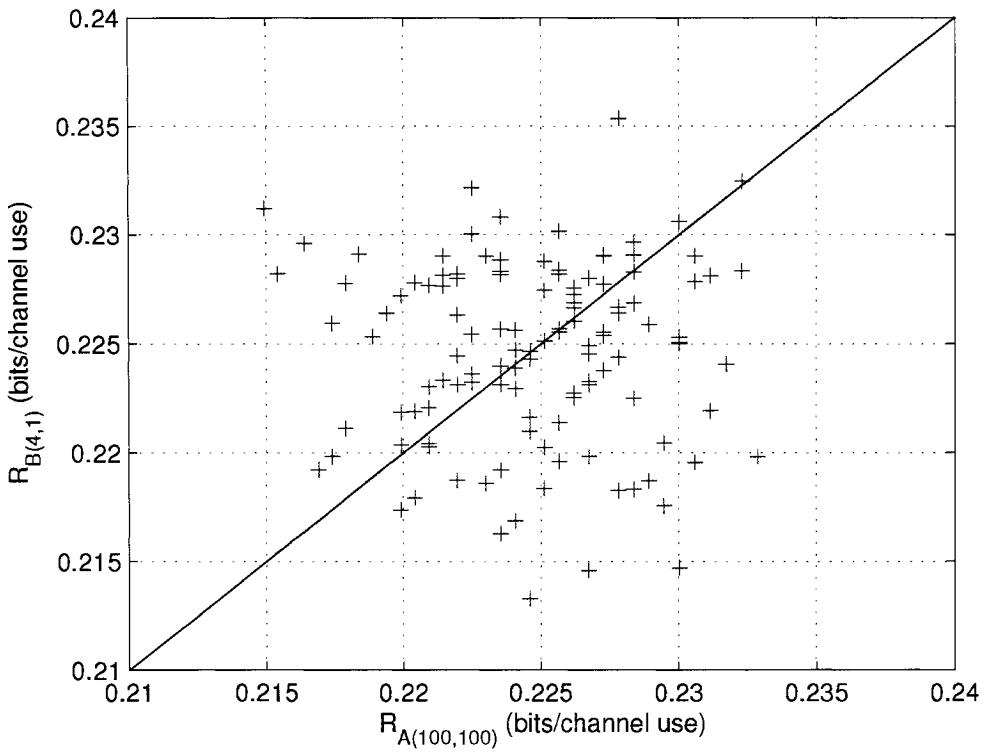

Figure 4.17: Scatter plot of realized rate by Algorithm $B(4,1)$ vs realized rate by Algorithm $A(100,100)$ over channel with capacity 0.25 bits/channel use.

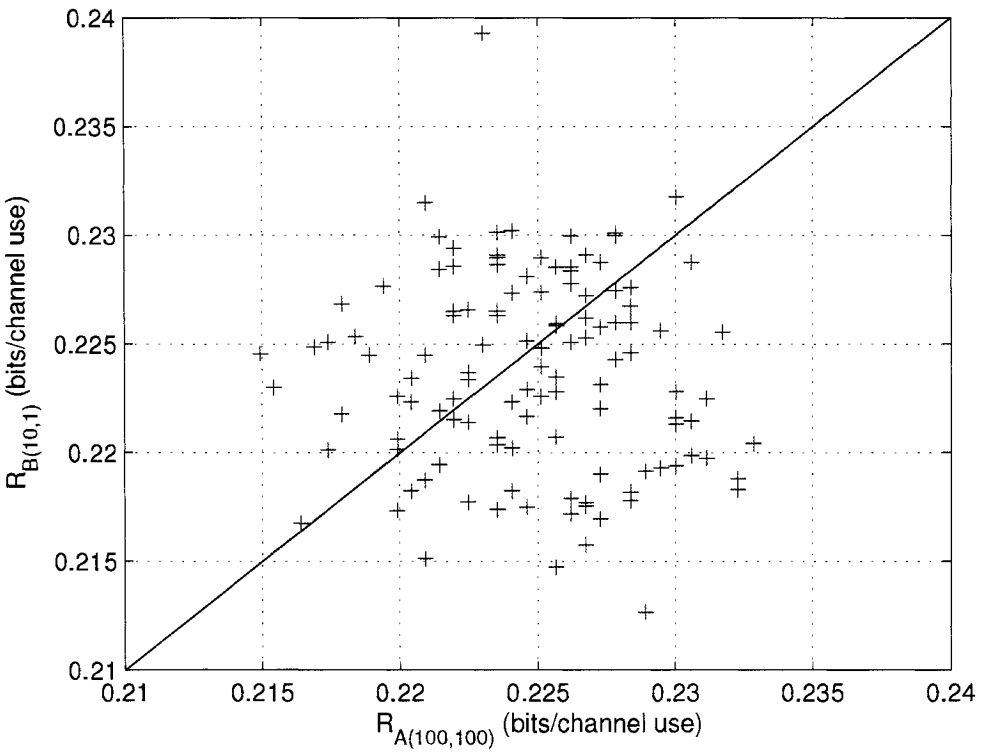

Figure 4.18: Scatter plot of realized rate by Algorithm $B(10, 1)$ vs realized rate by Algorithm $A(100, 100)$ over channel with capacity 0.25 bits/channel use.

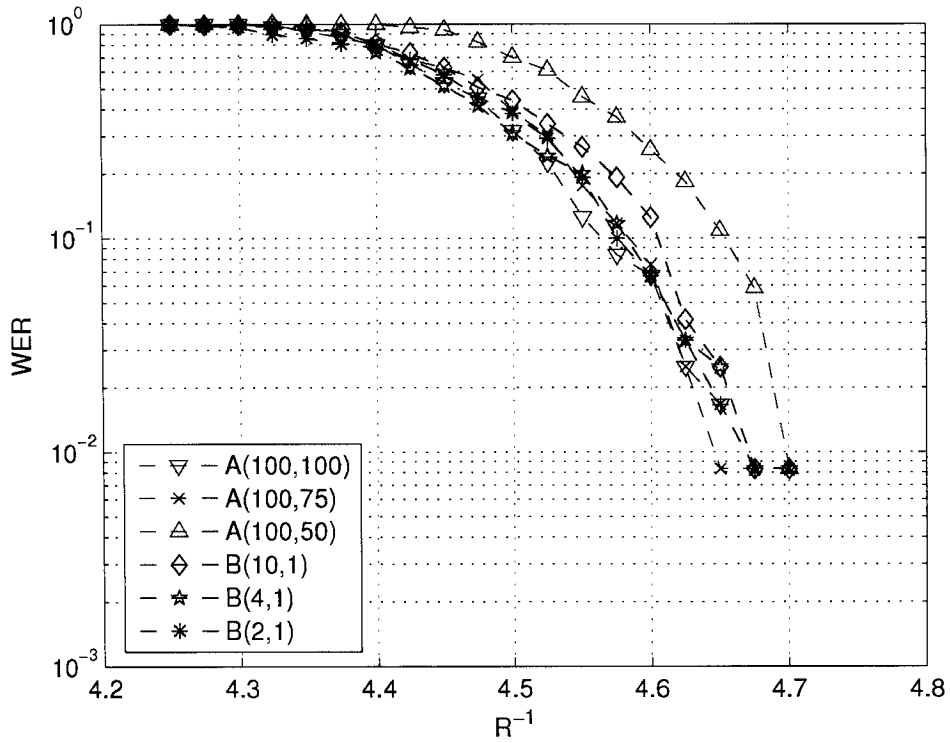

Figure 4.19: Word error rate as a function of $1/R$ for the two algorithms over channel with capacity 0.25 bits/channel use.

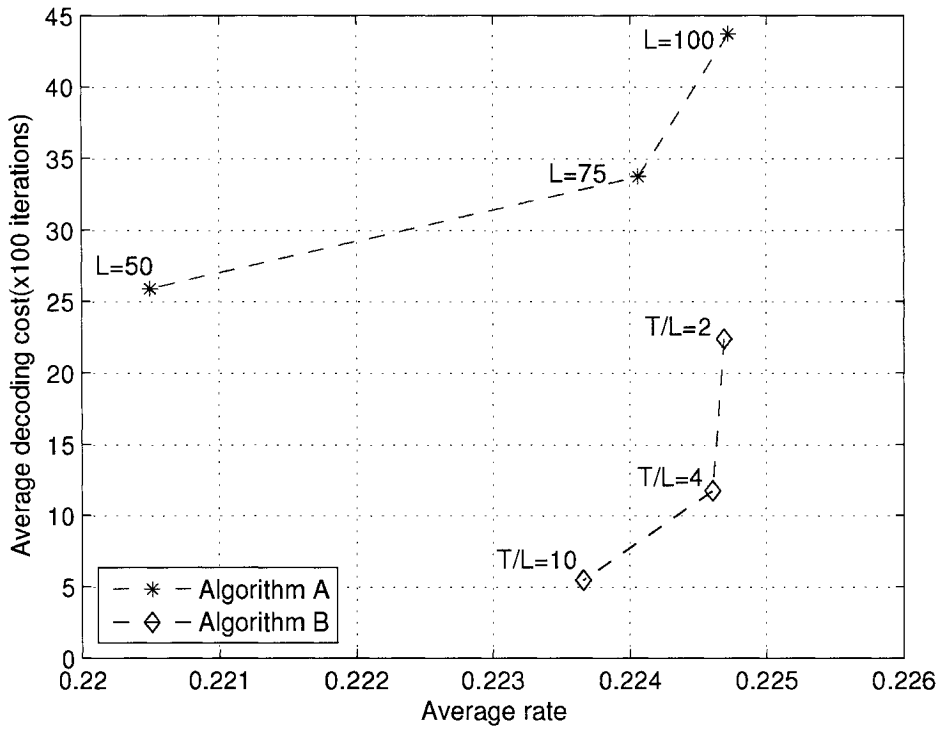

Figure 4.20: Average decoding cost vs average realized rates for the two algorithms over channel with capacity 0.25 bits/channel use.

From these figures, it is easily to see the Algorithm B take more advantages than Algorithm A in complexity reducing in terms of realized rate over channels with all different capacities. But there are also some slightly different among the different capacities. Next, we will discuss them respectively.

4.2.1 On channel with capacity 0.5 bits/channel use

In Figure 4.1, we see that most of the scatter points are below but close to the 45 degree line, which point represents the rate pair (R_x, R_y) (R_x means the rate corresponding to X axis, R_y means the rate corresponding to Y axis). If the amount of point "+" underneath the 45 degree line more than those above the 45 degree line, average realized rate corresponding to Algorithm $A(100, 100)$ is greater than average realized rate

corresponding to Algorithm $A(100, 75)$, thus, the performance of Algorithm $A(100, 100)$ is better than Algorithm $A(100, 75)$. Vice versa. Comparing Algorithm $A(100, 75)$ and Algorithm $A(100, 100)$ is reduced to compare the amount of points of above the 45 degree line and underneath the 45 degree line, namely, which side has more points, the algorithm corresponding to the side has better performance in terms of realized rate. This explanation holds on all further plots of performance comparison between any two algorithms. Obviously, the number of points underneath the 45 degree line is more than above the line in this figure, which indicates a small loss of performance in Algorithm $A(100, 75)$ comparing with Algorithm $A(100, 100)$. This suggests that as a performance benchmark, the parameter setting for Algorithm A should have $L > 75$.

Now, we are going to determine the benchmark of "L" in Algorithm A, (here, we assume the decoding interval T fixed to 100). The histogram of Algorithm $A(100, 100)$ is shown in Figure 4.2. This results is very close the histogram in [11]. We then use Algorithm $A(100, 100)$ as our benchmark for comparing with Algorithm B.

Figure 4.3 shows the performance comparison between Algorithm B and Algorithm A for same (T, L) setting. Definitely, Algorithm $B(100, 50)$ is better than Algorithm $A(100, 50)$ in our setting. This result gives the evidence support of Theorem II.

Figure 4.4 show that Algorithm $B(2, 1)$ has the same performance with Algorithm $B(20, 10)$, it verified that ratio T_B/L_B plays the primary role in decoding. So, Algorithm $B(2, 1)$ is identical to Algorithm $B(20, 10)$ in sense of realized rate, but the decoding delay of Algorithm $B(2, 1)$ is only $1/10$ of Algorithm $B(20, 10)$. Thus, we will use Algorithm $B(2/1)$ to represent the family Algorithm B with ratio $T/L = 2$.

Algorithm $B(2, 1)$ often outperforms Algorithm $A(100, 100)$ in terms of realized rate, as shown in Figure 4.5. We note that the average decoding cost of Algorithm $B(2, 1)$ is about 50% of Algorithm $A(100, 100)$, and the decoding delay of Algorithm $B(2, 1)$ is only 1% of Algorithm $A(100, 100)$.

This inspires us to keep maximizing the ratio T_B/L_B of the Algorithm B without big performance loss. We have tested Algorithm $B(4, 1)$ and Algorithm $B(6, 1)$ as shown in Figure 4.6 and Figure 4.7. The performance subsequently deteriorates apparently as the ratio T_B/L_B raised to $6/1$ in Algorithm B.

The word error rates for the two algorithms with various parameter settings are shown

in Figure 4.8. Our results suggest that Algorithms $B(20, 10), B(2, 1)$ give very similar word error performance, similar to that of $A(100, 100)$. This confirms that the ratio T/L acts the major function in the performance of Algorithm B.

Figure 4.9 shows that the trade-off between average decoding cost and average realized rate achieved with the two algorithms respectively. Significant improved trade-off is seen with Algorithm B. We explicitly note that with Algorithm B, one can always choose $L = 1$ thereby minimizing decoding delay. Depending on the performance that one is willing to sacrifice, comparing with Algorithm A, $50 - 80\%$ reduction of decoding cost is achievable with Algorithm B.

The comparison of the pair of (average realized rate, average decoding cost) between Algorithm A and Algorithm B in different parameters is presented in in Table 4.1.

	Algorithm A	Algorithm B
$(100, 100)$	$(0.4576, 1859)$	
$(100, 75)$	$(0.4543, 1508)$	
$(100, 50)$	$(0.4434, 1262)$	
$(2, 1)$		$(0.4584, 962)$
$(4, 1)$		$(0.4567, 550)$
$(6, 1)$		$(0.4551, 412)$
$(10, 1)$		$(0.4522, 301)$

Table 4.1: Comparison of the pair of (average realized rate, average decoding cost) between Algorithm A and Algorithm B in various settings.

4.2.2 On channel with capacity 0.75 bits/channel use

Similarly, we tested all possible parameter settings for Algorithm A to find a benchmark of iteration number (assuming T has been fixed to 100, same as on channel with capacity 0.5 bits/channel use). In Figure 4.10, we see that most of the scatter points are below the 45 degree line, which suggests Algorithm $A(100, 100)$ will be used as our benchmark.

After the parameters in Algorithm A have been fixed, we are now searching an optimal configurations of Algorithm B. The procedure has been demonstrated in Figure 4.11 and Figure 4.12.

The word error rates for the two algorithms with various parameter settings are shown in Figure 4.13. Algorithm $B(2,1)$ outperforms in word error probability comparing with Algorithm $A(100,100)$.

Figure 4.14 shows that the trade-off between average decoding cost and average realized rate achieved in the channel with capacity 0.75 bits/channel use. Again, Algorithm B demonstrated the significant improving over channels with high realized capacity. Algorithm $B(2,1)$ can reduce the complexity around 50% , and also improve the peformance in terms of realized rate comparing with Algorithm $A(100,100)$. Algorithm $B(10,1)$ can reduce the complexity up to 88% with sacrifice only 2% realized rate comparing with Algorithm $A(100,100)$.

The comparison of the pair of (average realized rate,average decoding cost) between Algorithm A and Algorithm B in different parameters is shown in Table 4.2.

	Algorithm A	Algorithm B
$(100,100)$	$(0.6358,7357)$	
$(100,75)$	$(0.6284,5692)$	
$(100,50)$	$(0.6099,4102)$	
$(2,1)$		$(0.6386,3684)$
$(4,1)$		$(0.6344,1925)$
$(10,1)$		$(0.6214,872)$

Table 4.2: Comparison of the pair of (average realized rate, average decoding cost) between Algorithm A and Algorithm B in various settings.

4.2.3 On channel with capacity 0.25 bits/channel use

In scenarios of channels with low capacity, following above two examples, we set Algorithm $A(100,100)$ as our benchmark for comparing with Algorithm B, as illustrated in Figure 4.15.

As observations from Figure 4.16 to Figure 4.18, the performance of Algorithm $B(2,1)$ and Algorithm $B(4,1)$ almost same as Algorithm $A(100,100)$. Even for Algorithm $B(10,1)$, the performance is still tolerable comparing with Algorithm $A(100,100)$.

Figure 4.19 shows the word error rates for the two algorithms with various parameter settings. Algorithm $B(2,1)$, $B(4,1)$, and $B(10,1)$ give close performance of word error

probability, and also close to Algorithm $A(100, 100)$. It confirms Algorithm $B(10, 1)$ has a good performance (realized rate) comparing with Algorithm $A(100, 100)$.

The trade-off between average decoding cost and average realized rate achieved with the two algorithms is shown in Figure 4.20. Comparing Algorithm $B(2, 1)$ with Algorithm $A(100, 100)$, the decoding cost can be reduced up to 48% without any lost of realized rate. Comparing Algorithm $B(4, 1)$ with Algorithm $A(100, 100)$, the decoding cost can be reduced up to 73% with sacrifice only 0.04% realized rate. Comparing Algorithm $B(10, 1)$ with Algorithm $A(100, 100)$, the decoding cost can be reduced up to 87% with sacrifice only 0.4% realized rate. Observing the trade-off figure, we suggest the Algorithm $B(4, 1)$ is perhaps the "optimal" decoding algorithm on the channel with low capacity in our setting.

The comparison of the pair of (average realized rate, average decoding cost) by Algorithm A and Algorithm B in different parameters is illustrated in Table 4.3.

	Algorithm A	Algorithm B
$(100, 100)$	$(0.2247, 4376)$	
$(100, 75)$	$(0.2241, 3375)$	
$(100, 50)$	$(0.2205, 2592)$	
$(2, 1)$		$(0.2247, 2241)$
$(4, 1)$		$(0.2246, 1174)$
$(10, 1)$		$(0.2237, 548)$

Table 4.3: Comparison of the pair of (average realized rate, average decoding cost) between Algorithm A and Algorithm B in various settings.

4.3 Summary

From above discussion, it can be easily seen that with no rate loss and even with rate gain, Algorithm B significantly improves upon Algorithm A in complexity. The complexity reduction achieved with Algorithm B ranges from 50% to 90% in terms of decoding cost, and is equal to about 1% in terms of decoding delay.

Across channels, it appears that the advantage of Algorithm B over Algorithm A is particularly significant on low-capacity channels.

- 0.5 bits/channel use:
  - Algo $B(2,1)$: cost: $\downarrow$ 50%, rate: no lost.
- 0.75 bits/channel use:
  - Algo $B(2,1)$: cost: $\downarrow$ 50%, rate: slightly $\uparrow$.
  - Algo $B(10,1)$: cost: $\downarrow$ 88%, rate: only $\downarrow$ 2%.
- 0.25 bits/channel use:
  - Algo $B(2,1)$: cost: $\downarrow$ 49%, rate: no lost.
  - Algo $B(4,1)$: cost: $\downarrow$ 73%, rate: only $\downarrow$ 0.04%.
  - Algo $B(10,1)$: cost: $\downarrow$ 88%, rate: only $\downarrow$ 0.4%.
- Summary:
  - Algo $B \downarrow 50-90\%$ decoding cost.
  - Algo $B \downarrow 99\%$ decoding delay.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Rateless coding are promising new techniques for communicating over channel uncertainty, particularly suited for wireless channels. However the decoding complexity, prior to this work, is relatively high comparing to fixed-rate codes. This limits its potential use in reality.

In this work, we present a simple modified decoding algorithm for communication over fading channels using Raptor codes. We show that the presented algorithm, at no cost of performance, offers significantly reduced complexity over the existing algorithms. Some theoretical bounds are also presented to shed some light on the behavior of the algorithm.

Equipped with this new algorithm, Raptor codes are becoming more competitive as a solution for communication over wireless channels.

5.2 Future work

As we discussed above, Algorithm B can reduce decoding complexity significantly. But, the current encoding method is not optimal for given the decoding algorithm – Algorithm B. It bring us a challenge to design a new optimal coder for the Algorithm B. We believe this future work will be very interesting and challenging.

References

[1] M. Luby, "LT codes," in *43rd Annual IEEE Symp. on the Found. of Comp. Sci.*, 2002, pp. 271–280.

[2] A. Shokrollahi, "Raptor codes," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2004, p. 36.

[3] A. Lapidoth and P. Narayan, "Reliable communication under channel uncertainty," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2148–2175, Oct. 1998.

[4] M. Medard, "The effect upon channel capacity in wireless communications of perfect and imperfect knowledge of the channel," *IEEE Trans. Inform. Theory*, vol. 46, no. 3, pp. 933–946, May 2000.

[5] S. C. Draper, B. J. Frey, and F. R. Kschischang, "Efficient variable length channel coding for unknown DMCs," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2004, p. 379.

[6] J. Castura, Y. Mao, and S. C. Draper, "On rateless coding over fading channels with delay constraints," submitted to IEEE ISIT 2006.

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

[8] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[9] D. Tse and P. Viswanath, *Fundamentals of wireless communication.* Cambridge University Press, June 2005.

[10] O. Etesami, M. Molkaraie, and A. Shokrollahi, "Raptor codes on symmetric channels," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2004, p. 38.

[11] R. Palanki and J. S. Yedidia, "Rateless codes on noisy channels," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2004, p. 37.

[12] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[13] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," *Eur.Trans. Telecomm.*, vol. 6, pp. 513–525, Sep./Oct. 1995.

[14] G. D. Forney, Jr., "On iterative decoding and the two-way algorithm," *Int. Symp. on Turbo Codes and Related Topics, Brest, France*, pp. 12–25, Sep. 1997.

[15] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, pp. 325–343, Mar. 2000.

[16] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregualr low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[17] Y. Mao and A. H. Banihashemi, "Decoding low-density parity-check codes with probabilistic scheduling," *IEEE Comm. Lett.*, vol. 5, no. 10, pp. 414–416, Oct. 2001.

[18] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commumn.*, vol. 16, pp. 219–230, Feb. 1998.

[19] H. Xiao and A. H. Banihashemi, "Graph-based message-passing schedules for decoding ldpc codes," *IEEE Trans. Comm.*, vol. 52, no. 12, pp. 2098–2105, Dec. 2004.

[20] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in gaussina graphical models of arbitrary topology," *Neural Computation*, vol. 13, pp. 2173–2200, 2001.

[21] R. G. Gallager, *Low Density Parity Check Codes.* Cambridge, MA: MIT Press, 1963.

[22] D. J. C. Mackay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *IEE Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[23] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under messages-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2000.

[24] J. G. Proakis, *Digital Communications, Fourth Edition.* MCGraw-Hill College, Aug. 2000.

[25] T. Etzion, A. Trachtenbeg, and A. Vardy, "Which codes have cycle-free tanner graphs?" *IEEE Trans. Inform. Theory*, vol. 45, pp. 2173–2181, Sep. 1999.

[26] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, Jul. and Oct. 1948.

[27] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the shnnaon limit," *IEEE Commun. Letter*, vol. 5, pp. 58–60, Feb. 2001.