Iterative Decoding

and

Graphical Code Representations

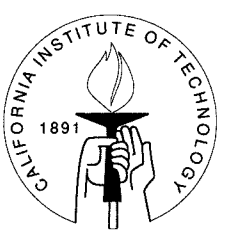
Thesis by

Meina Xu

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

1999

(Submitted May 21, 1999)

Acknowledgements

No words can express my appreciation and gratitude to my advisor Professor Robert J. McEliece for giving me an opportunity to study at Caltech and for providing me with the guidance and encouragement during the course of this work. I have benefited tremendously from his knowledge and insight of science. His many suggestions and ideas have often led to exciting approaches to problems, and his enthusiasm on research has greatly influenced mine. I am greatly indebted to him for his unconditional support, unlimited patience and encouragement, especially on helping me get back to research after medical leaves for recovering from an auto accident and the follow-up surgeries. This thesis would not have been written without him not giving up on me.

My special thanks go to Professor Shu Lin for introducing me to the field of coding and giving me for many invaluable advice and counsel. His great knowledge of the field has been a tremendous resource for me.

I am indebted to Srinivas Aji for reading the manuscripts and providing constructive criticism. Srinivas' knowledge of the generalized distributive law, iterative decoding, mathematics and other subjects has been a great resource for me, and I am also grateful to him for his friendship and support.

I am grateful to Professor Michael Tanner for the many fruitful and engaged discussions on iterative decoding, and Professor Anthony Kuh for his encouragement and friendship. I am also grateful to Professor P.P. Vaidyanathan, Jehoshua Bruck, Dariush Divsala, Joel Franklin, Michael Tanner, and Ghassan Kawas Kaleh for their interest in my work, their helpful comments, and for spending some of their precious time serving on my candidacy/defense committees.

I would like to thank Lilian Porter, Linda Dozsa and all the department secretaries for their assistance and help on administrative matters and their care and kindness towards me. I also would like to thank our former and current system administrators

Robert Freeman and Joseph Chiu for keeping the computers up and running, and their availability even in the late night hours.

I am thankful to my former and current group-mates: Srinivas Aji, Mohamed-Slim Alouini, Aamod Khandekar, Jung-Fu Cheng, Gavin Horn, Syed Ali Jafar, Hui Jin, Lifang Li, Wei Lin, Kim Lumbard, Neelesh B. Mehta, Sriram Vishwanath, and Zhong Yu for their friendship and many fruitful discussions. I would like to thank my good friends Sony Akkarakaran and Murat Mese in the DSP group, Vincent Bohossian, Paul LeMahieu, Charles Fan, and Lihao Xu in the parallel and distributed computing group, and Song Yong in the Vision group for hosting me in their labs, and friendship.

I am very grateful to my special friend Kathleen Bartle-Schulweis in the women's center for her support and love, especially when I returned to Caltech after a leave of absence due to an auto accident.

I would like to thank my friends Cecilia Chu, Martin Han, Susan Huang, Michael Lee, Nhat Nguyen, Darryl Tamura, Jeff Tse, Anthony Uy, Jorge Wong, and Mai Yong for their support and for the many enjoyable moments we spent together.

I owe a very special thank you to Su-Chin Chou for the days and nights she slept at my bedside and cared for me after my release from hospitalization due to the auto accident, and to Chuan-Cheng Cheng, Shao-Ching Hung, Hui-Ming Hung, Chao-Ping Hsu, Yu-Ting Kao, and Hsi-I Yang for the countless hours which they have spent helping me to re-learn walking, taking me to doctors, giving me counsel and support, and, over all, being my family away from home.

I would like to thank my uncles Kwai-Yin Lum and Chin Lum, my cousins Wendy, Lisa, Kenneth, and Chi-Hon, for letting me and my families to stay in their home, for helping us settle in the United States, and for their encouragement and support.

Last, but certainly not the least, I would like to express my greatest gratitude to my parents and my brothers Sam and Ken for their sacrifice and support. They have always believed in me and encouraged me every step of the way.

Abstract

Since the invention of turbo codes, there has been an explosion of interest in iterative decoding and graphical representation of codes. This thesis examines the iterative decoding of codes defined on graphs with cycles, which appears to be an efficient means of achieving the Shannon limit. Much of this analysis is on the iterative min-sum decoding of tail-biting codes and cycle codes. We have identified the pseudocodeword as the cause of the suboptimal performance of the iterative decoder, and we have obtained a union bound for the performance of the iterative decoder on both AWGN and BSC channels. Using the union bound argument, for cycle codes, we have shown that the performance of the iterative decoder is asymptotically as good as that of the ML decoder. As for tail-biting codes, the same thing is true if the lowest weight pseudocodeword is at least the minimum weight of the code. Unfortunately, the analysis of tail-biting codes and cycle codes does not extend to turbo codes and low density parity check codes in general. Our next approach is to determine the average behavior of message passing algorithms by studying the evolution of their "message" densities. For the class of "repeat and accumulate" serially concatenated turbo-like codes, we have devised an algorithm for determining their "threshold" values. When the signal-to-noise ratio is larger than the threshold value, the error probabilities of message passing algorithms approach zero, whereas if the signal-to-noise ratio is less than the threshold value, the error probability stays bounded away from zero. Some message passing algorithms and graphical representation of codes are efficient means of devising ML or MAP decoding algorithms. We have proposed a junction tree representation for linear block codes, and we have shown that the minimum junction tree can be less complex than the minimal trellis.

Contents

List of Figures

Chapter 1 Introduction

A fundamental problem in communication is how to achieve reliable communication efficiently over unreliable channels. The basic structure of a communication system [55, 63] is illustrated in Figure 1.1. The message to be sent is encoded before it is transmitted on the channel. At the receiving end, the decoder finds an estimate of the message from the channel output.

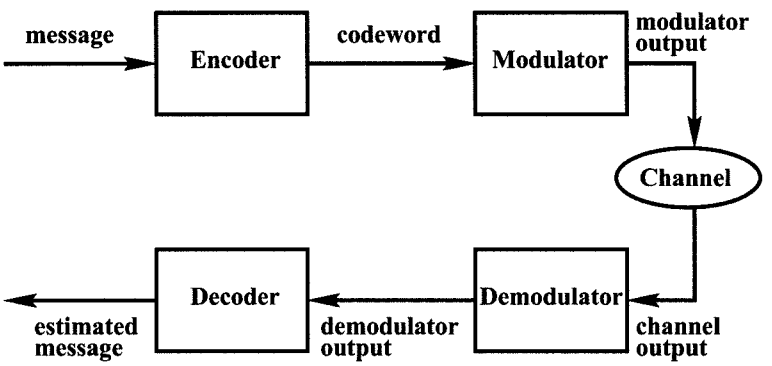


Figure 1.1: A communication system.

Shannon's channel coding theorem states that reliable communication can be achievable as long as the rate of a code does not exceed the "capacity" of the channel. Though random codes with large block length perform close to the Shannon limit [40, 60, 67], maximum a posteriori (MAP) or maximum likelihood (ML) decoding of such codes is computationally infeasible. Traditionally[1], code design has been addressed by developing codes with a lot of structure, which lend themselves to feasible decoding [39, 34], but these codes are far from the Shannon limit.

The discovery of turbo codes revolutionized the field of coding by introducing an iterative decoding algorithm which efficiently decodes long pseudo-random codes and thus bridges the gap between the Shannon limit and the practically achievable values.

[1] A notable exception is Gallager's low density parity check codes.

In part of this thesis, we will examine the iterative decoding of codes defined on graphs with cycles, which appears to be an efficient means of achieving the Shannon limit. Much of the analysis will be on the iterative min-sum decoding of tail-biting codes and cycle codes, and we will provide a theoretical explanation for the good performance of iterative decoding on these codes.

1.1 Iterative Decoding

Since the invention of turbo codes by Berrou, Glavieux, and Thitimajshima in 1993 [8], there has been an explosion of interest in iterative decoding and graphical representation of codes. At the same time, there has also been a re-awakened interest in the long forgotten low density parity check (LDPC) codes introduced by Gallager [25] in 1963.

Turbo codes and LDPC codes can be represented by cycle free graphs. In this case iterative decoding boils down to the forward-backward algorithm or Viterbi's algorithm, and results in a MAP or ML decision. However, the computation involved is infeasible.

For computationally feasible decoding, the best graphical representations of turbo and LDPC codes contain cycles. It is well known that iterative decoding on graphs with cycles is suboptimal[2]. Still, there is a large body of empirical evidence illustrating the excellent performance of iterative decoding on graphs with cycles. In fact, in the coding community, there has been a race on designing turbo codes and LDPC codes which perform closer and closer to the Shannon limit, and the best known LDPC code is only 0.06 dB from the Shannon limit [58].

What is still lacking, however, is a satisfactory theoretical explanation of why iterative decoding algorithms perform as well as they do. The major goal behind this thesis has been to search for such a theoretical explanation.

[2]The decoding does not result in a ML or MAP decision.

1.2 Message Passing Algorithms on Graphs

Graphical representations of codes are not only useful as code descriptions, but most importantly, they lead to natural decoding algorithms when the graphs are considered as communication networks in which "messages" between vertices are communicated through the edges connecting them. While Viterbi's algorithm is the best known decoding algorithm on a trellis, other trellis decoding algorithms are the BCJR algorithm (also known as the forward-backward algorithm), and sequential decoding [24, 20].

Gallager's decoding algorithm for LDPC codes is the first message passing algorithm. Tanner generalized Gallager's LDPC codes to codes on general bipartite graphs, called Tanner graphs [64]. More recently, Wiberg et al. extended Tanner graphs to Tanner-Wiberg-Loeliger graphs which include state nodes [69, 70]. Recent developments on graphical representations includes Bayesian networks [53, 42], junction graphs [1, 3], and factor graphs [23].

Message passing algorithms used on the above mentioned graphs include the Gallager-Tanner-Wiberg decoding algorithm (sum-product and min-sum algorithms), Pearl's belief propagation algorithm, and the generalized distributive law (GDL). Viterbi's algorithm and the forward-backward algorithm are special cases of all these algorithms when the underlying graphs are cycle free. The GDL includes as special cases Pearl's belief propagation algorithm and the Gallager-Tanner-Wiberg decoding algorithm.

In this thesis, we will study message passing algorithms on graphs which represent tail-biting codes, cycle codes, and a class of serially concatenated turbo-like codes called "repeat and accumulate" codes.

In the course of studying message passing algorithms and graphical representation of codes, we came to discover that some message passing algorithms and graphical representation of codes are efficient means of devising exact (ML or MAP) decoding algorithms. This study is also be reported in the thesis.

1.3 Thesis Outline

In Chapter 2, we briefly introduce the Generalized Distributive Law [1, 3], which forms the basis of the results in Chapters 3 and 4. We also present junction graphs for tail-biting codes, and we treat a Tanner graph as a special junction graph. Our view of message passing algorithms, specifically in Chapters 5, 6, and 7, will be as an instance of the GDL, and junction graphs are the graphical representations in this thesis.

Chapters 3 and 4 are applications of the GDL. The GDL gets its efficiency by reordering certain computations using the distributive law, so we will apply the GDL to reduce decoding complexity in these two Chapters. In Chapter 3, we will introduce a junction tree representation for linear block codes, and we will show that the minimum complexity junction tree can be less complex than the minimal trellis. In Chapter 4, we will view the problem of decoding a tail-biting trellis as a matrix chain multiplication problem in the GDL framework, which can be represented as a single cycle junction graph. By viewing Viterbi's algorithm as an algorithm for multiplying a chain of matrices, we will prove the optimality of the algorithm. The material in these two chapters has been presented earlier in [43, 44, 45].

Chapter 5 and 6 are devoted to the study of iterative min-sum decoding of tail-biting codes and cycle codes. We investigate the degradation due to iterative decoding, and we obtain a union bound to quantify the performance loss. We examine the "pseudodistance" of these codes to further quantify the degradation. The work in Chapter 5 has been presented earlier in [2, 4].

In Chapter 7, we analyze message passing algorithms for a special class of serially concatenated turbo-like codes called "repeat and accumulate codes." We will present an algorithm to determine the "threshold" value. If the signal-to-noise ratio is above the threshold, the belief propagation decoding error probability approaches zero exponentially fast in the length of the codes; whereas for signal-to-noise ratio below the threshold, the error probability stays bounded away from zero.

Chapter 8 contains our concluding remarks.

Chapter 2 The Generalized Distributive Law

There has been a great interest in message passing algorithms on graphs in the coding community. Message passing algorithms include Viterbi's algorithm, the forward-backward algorithm, the Gallager-Tanner-Wiberg decoding algorithm (sum-product or min-sum algorithm), Pearl's belief propagation algorithm, and the generalized distributive law (GDL). The GDL is a generalization of all these algorithms.

In this thesis, our view of message passing algorithm is the GDL applied to a junction graph. In this chapter, we will use Aji and McEliece's [1, 3] notation to introduce the GDL and junction graphs, and we will also treat a Tanner graph as a special type of junction graph.

2.1 The Marginalizing a Product Function Problems

In this section, we will describe the "marginalizing a product function" (MPF) problem, which is a general computational problem solved by the GDL. The GDL can greatly reduce the number of "additions" and "multiplications" in MPF problems. In a MPF problem, the notions of "additions" and "multiplications" are generalized to a "commutative semiring."

Definition. *A* commutative semiring $(K, +, \cdot)$ *is a set* K*, together with two binary operations called "*$+$*" and "*$\cdot$*", which satisfy the following:*

A1. The operation "$+$*" is associative and commutative, and there is an additive identity element called "*0*" such that* $k + 0 = k$ *for all* $k \in K$*.*

A2. The operation "$\cdot$*" is associative and commutative, and there is a multiplicative*

identity element called "1" such that $k \cdot 1 = k$ *for all* $k \in K$.

A3. The distributive law *holds, i.e.,* $(a \cdot b) + (a \cdot c) = a \cdot (b + c)$, *for all triples* (a, b, c) *from* K.

The difference between a semiring and a ring is that in a ring, additive inverses are required, i.e., $(K, +)$ are required to be a group in a ring. Thus every commutative ring is automatically a commutative semiring. The two commutative semirings which we use in this thesis are given in Example 2.1.1 and Example 2.1.2.

Example 2.1.1. *Commutative semiring* $(\mathbb{R}^+, +, \cdot)$, *where* $\mathbb{R}^+$ *is the set of nonnegative real numbers. Here the operation "+" is the ordinary addition with the real number* 0 *as its identity, and the operation "·" is the ordinary multiplication with the real number* 1 *as its identity. We will call this the* sum-product *semiring.*

Example 2.1.2. *Commutative semiring* $(\mathbb{R} \cup \{\infty\}, min, +)$. *Here the operation "*min*" is defined as the operation of* taking the minimum . *We define* $\min(k, \infty) = k$, *for all* $k \in \mathbb{R} \cup \{\infty\}$, *i.e.,* ∞ *is the identity for the operation "*min.*" The operation "+" is the ordinary addition with the real number* 0 *as its identity. The distributive law holds because for any* $a, b, c \in \mathbb{R} \cup \{\infty\}$,

$$\min(a + b, a + c) = a + \min(b, c)$$

We will call this the min-sum *semiring.*

We now introduce the "marginalizing a product function" (MPF) problem.

Let $x_1, \cdots, x_n$ be variables taking values in the finite alphabets $A_1, \cdots, A_n$. Let $S_i = \{i_1, \cdots, i_k\}$ be a subset of $\{1, \cdots, n\}$. Let A_{S_i} represent the Cartesian product $A_{i_1} \times \cdots \times A_{i_k}$. Let x_{S_i} represent the variable list $x_{i_1}, \cdots, x_{i_k}$. The variable list $x_1, \cdots, x_n$ is denoted by $\mathbf{x}$, and the product $A_{\{1,\cdots,n\}}$ by $\mathbf{A}$.

Now let $\mathcal{S} = \{S_1, \cdots, S_M\}$ be M subsets of $\{1, \cdots, n\}$. For each i, there is a function $\alpha_i : A_{S_i} \to R$, where R is a commutative semiring. We call the variable lists x_{S_i} the *local domains* and the functions α_i the *local kernels*. Define the *global kernel*

$\beta : \mathbf{A} \to R$ as:

$$\beta(x_1, \cdots, x_n) = \prod_{i=1}^{M} \alpha_i(x_{S_i}). \tag{2.1}$$

The MPF problem is to compute the *S_i-marginalization* of the product of the global function for one or more of the indices $i \in \{1, \cdots, M\}$, i.e., the function $\beta_i : A_{S_i} \to R$, where

$$\beta_i(x_{S_i}) = \sum_{x_{S_i^c} \in A_{S_i^c}} \beta(x), \tag{2.2}$$

where S_i^c is the complement of the set S_i relative to the "universal set" $\{1, \cdots, n\}$.

2.2 The Decoding Problem as an Instance of the Marginalizing a Product Function Problem

Consider a communication system as shown in Figure 2.1. An information block $U = (u_1, \cdots, u_k)$ of length k is encoded into $X = (x_1, \cdots, x_n)$ which is then transmitted over the channel. From the channel output $Y = (y_1, \cdots, y_n)$, the decoder computes an estimate $\hat{U}$ of U.

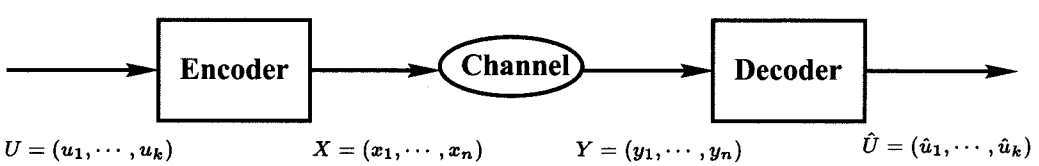


Figure 2.1: A communication system.

We assume that each symbol in U is generated independently, so $p(U) = \prod_{i=1}^{k} p(u_k)$. Each U is encoded to X according to a deterministic encoding rule, so $p(X|U)$ is 1 if X is the encoded version of U and 0 otherwise. We also assume the channel is memoryless, i.e., $p(Y|X) = \prod_{j=1}^{n} p(y_i|x_i)$. Then the joint probability $p(U, X, Y)$ can

be written as:

$$p(U,X,Y) = p(U)p(X|U)p(Y|X) \tag{2.3}$$

$$= \prod_{i=1}^{k} p(u_i)p(X|U) \prod_{j=1}^{n} p(y_j|x_j). \tag{2.4}$$

Let $\phi(X)$ be a function proportional to the probability that X is transmitted, so that $\phi(X) = 0$ if X is not a codeword. If we assume each codeword is equally likely, then $\phi(X)$ can be considered as the indicator function of the set of codewords. Since X is a codeword if and only if it satisfies a set of l conditions (see Example 2.2.2), then

$$\phi(X) = \prod_{j=1}^{l} \phi_j(X). \tag{2.5}$$

A MAP decoder finds $\hat{u}_i$ or $\hat{x}_i$ such that:

$$\hat{u}_i = \arg \max_{u_i} p(u_i|Y) = \arg \max_{u_i} \sum_{\{u_j\}j\neq i} p(U,X,Y), \tag{2.6}$$

$$\hat{x}_i = \arg \max_{x_i} \sum_{\{x_j\}j\neq i} p(Y|X)\phi(X). \tag{2.7}$$

A ML decoder finds $\hat{u}_i$ or $\hat{x}_i$ such that:

$$\hat{u}_i = \arg \max_{u_i} p(U|Y) = \arg \max_{u_i} \max_{\{u_j\}j\neq i} p(U,X,Y), \tag{2.8}$$

$$\hat{x}_i = \arg \max_{x_i} \max_{\{x_j\}j\neq i} p(Y|X)\phi(X). \tag{2.9}$$

If we take the negative of the logarithm of equations (2.8) and (2.9), both the MAP and ML decoding problems can be unified as one MPF problem in the sum-product and min-sum semiring.

In this thesis, depending on which commutative semiring is used, we will call the GDL the sum-product or the min-sum algorithm respectively.

We will conclude this section with two examples.

Example 2.2.1. *(Calderbank et al. [14]) Consider the generator matrix which gives the minimal tail-biting trellis for the* $(8,4,4)$ *extended Hamming code, viz.,*

$$G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}. \tag{2.10}$$

We would like to find the most probable $(\hat{u}_1, \cdots, \hat{u}_4)$*, i.e.,*

$$\hat{u}_i = \arg \min_{u_i} \min_{\{u_j\} j \neq i} \left[\left(\sum_j p(u_j) \right) \left(\sum_j (-\log(p(y_j|x_j)) + \chi'(X|U)) \right) \right], \tag{2.11}$$

for $i = 1, \cdots, 4$*. In this example,* $\chi'(X|U)$ *is*

$$\begin{aligned} \chi'(X|U) = {} & \chi(x_1 + u_1) + \chi(x_2 + u_1 + u_4) + \chi(x_3 + u_1 + u_2 + u_4) + \chi(x_4 + u_1 + u_2) \\ & + \chi(x_5 + u_3) + \chi(x_6 + u_2 + u_3) + \chi(x_7 + u_2 + u_3 + u_4) + \chi(x_8 + u_3 + u_4), \end{aligned}$$

where χ *is* 0 *if its argument sum (mod 2) is zero and* ∞ *otherwise.*

Equation 2.11 is a MPF problem in the min-sum semiring with the following local domains and local kernels.

local domain	*local kernel*
$\{u_1\}$	1
$\vdots$	$\vdots$
$\{u_4\}$	1
$\{x_1, u_1, u_4\}$	$-\log p(y_1\|x_1) - \log p(u_1) + \chi(x_1 + u_1)$
$\{x_2, u_1, u_4\}$	$-\log p(y_2\|x_2) - \log p(u_4) + \chi(x_2 + u_1 + u_4)$
$\{x_3, u_1, u_2, u_4\}$	$-\log p(y_3\|x_3) + \chi(x_3 + u_1 + u_2 + u_4)$
$\{x_4, u_1, u_2\}$	$-\log p(y_4\|x_4) - \log p(u_2) + \chi(x_4 + u_1 + u_2)$
$\{x_5, u_3\}$	$-\log p(y_5\|x_5) + \chi(x_5 + u_3)$
$\{x_6, u_2, u_3\}$	$-\log p(y_6\|x_6) + \chi(x_6 + u_2 + u_3)$
$\{x_7, u_2, u_3, u_4\}$	$-\log p(y_7\|x_7) + \chi(x_7 + u_2 + u_3 + u_4)$
$\{x_8, u_3, u_4\}$	$-\log p(y_8\|x_8) - \log p(u_4) + \chi(x_8 + u_3 + u_4)$

Example 2.2.2. *Consider the* $(9, 4, 3)$ *cycle code with parity check matrix*

$$H = \begin{array}{c} \begin{array}{ccccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}. \tag{2.12}$$

We would like to find the most probable codeword $(\hat{x}_1, \cdots, \hat{x}_9)$*, i.e.,*

$$\hat{x}_i = \arg \min_{x_i} \min_{\{x_j\} j \neq i} \sum_{j=1}^{9} \left[-\log(p(y_j|x_j)) + \phi'(X) \right], \tag{2.13}$$

for $i = 1, \cdots, 9$*, where* $\phi'(X)$ *is* 0 *if* X *is a codeword and* ∞ *otherwise. In this*

example, $\phi'(X)$ *is*

$$\phi'(X) = \chi(x_1 + x_2 + x_8) + \chi(x_2 + x_3 + x_5) + \chi(x_3 + x_4 + x_6) \\ + \chi(x_1 + x_4 + x_9) + \chi(x_7 + x_8 + x_9) + \chi(x_5 + x_6 + x_7),$$

where χ *is* 0 *if its argument sum (mod 2) is zero and* ∞ *otherwise.*

Equation 2.13 is a MPF problem in the min-sum semiring with the following local domains and local kernels.

local domain	*local kernel*
$\{x_1\}$	$-\log p(y_1\|x_1)$
$\vdots$	$\vdots$
$\{x_9\}$	$-\log p(y_9\|x_9)$
$\{x_1, x_2, x_8\}$	$\chi(x_1 + x_2 + x_8)$
$\{x_2, x_3, x_5\}$	$\chi(x_2 + x_3 + x_5)$
$\{x_3, x_4, x_6\}$	$\chi(x_3 + x_4 + x_6)$
$\{x_1, x_4, x_9\}$	$\chi(x_1 + x_4 + x_9)$
$\{x_7, x_8, x_9\}$	$\chi(x_7 + x_8 + x_9)$
$\{x_5, x_6, x_7\}$	$\chi(x_5 + x_6 + x_7)$

2.3 The Generalized Distributive Law

In many cases, the GDL provides an algorithm to compute MPF problems more efficiently than by direct computation, and it gets its efficiency by reordering the computation using the distributive law.

Definition. *A* junction tree *is a tree in which the subgraph induced by the vertices whose local domains contain any given element is connected.*

When the vertex labels given by the local domains S_i, for $i \in \{1, \cdots, M\}$, form a junction tree, the GDL gives the exact solution to the marginals β_i.

Let $(v_1, \cdots, v_M)$ denote the vertex labels given by the local domains $(S_1, \cdots, S_M)$. Denote the edge connects v_i and v_j by $v_i \,\mathrm{adj}\, v_j$. Then the "message" from v_i to v_j is

a table containing the value of a function $\mu_{i,j} : A_{S_i \cap S_j} \to R$:

$$\mu_{i,j}(x_{S_i \cap S_j}) = \sum_{x_{S_i \setminus S_j} \in A_{S_i \setminus S_j}} \alpha(x_{S_i}) \prod_{\substack{v_k \text{ adj } v_i \\ k \neq j}} \mu_{k,i}(x_{S_k \cap S_i}). \tag{2.14}$$

Initially, all $\mu_{i,j}$ are defined to be the multiplicative identity of the semiring. When v_i wishes to send a message to v_j, it is computed by multiplying its local kernel and the messages it received from all its neighbors except v_j and then marginalizing out all the variables not in S_j.

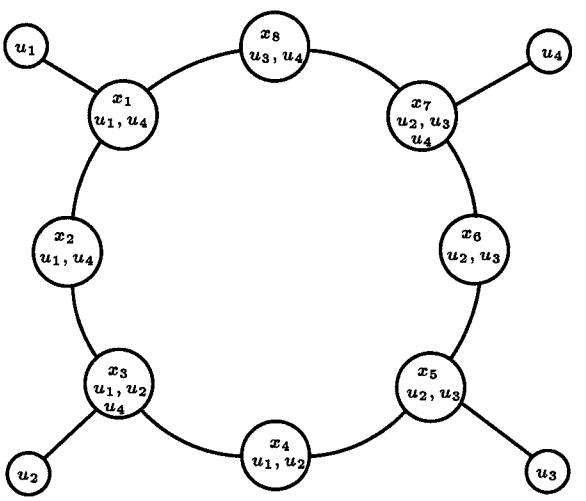


Figure 2.2: The junction graph for the $(8, 4, 4)$ tail-biting Hamming code.

When v_i has received messages from all its neighbors, it can compute its "state," which is a table containing values of a function $\sigma_i : A_{S_i} \to R$ as :

$$\sigma_i(x_{S_i}) = \alpha(x_{S_i}) \prod_{v_k \text{ adj } v_i} \mu_{k,i}(x_{S_k \cap S_i}). \tag{2.15}$$

The state σ_i is the required β_i.

When the local domains do not form a junction tree, we will ignore the cycles and still apply the updating rule to the junction graph[1] anyway. The single cycle junction

[0]A *junction graph* is a graph in which the subgraph induced by the vertices whose local domains contain any given element is connected.

graph shown in Figure 2.2 is the junction graph formed from the local domains of the $(8, 4, 4)$ tail-biting code in Example 2.2.1. Figure 2.3 shows the junction graph for the $(9, 4, 3)$ cycle code in Example 2.2.2, and this junction graph can be viewed as a Tanner graph[2].

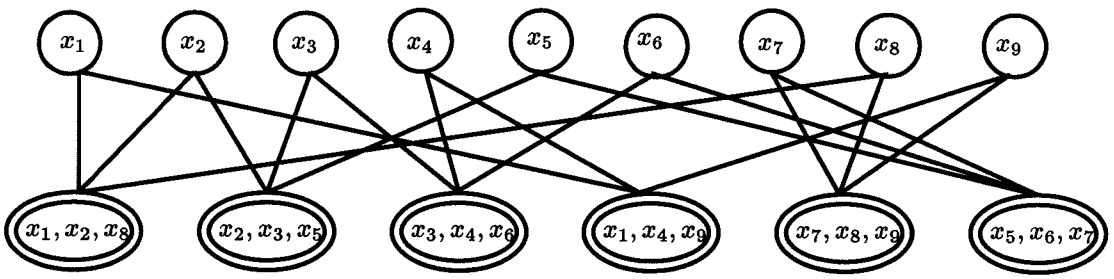


Figure 2.3: The junction graph for the $(9, 4, 3)$ cycle code in Example 2.2.2. The nodes in the bottom row act as check nodes, so this junction graph is a Tanner graph.

[2]A Tanner graph $G = (V, E)$ is a bipartite graph consisting of a set of codeword nodes V_m and check nodes V_c, where $V = V_m \cup V_c$ with $V_m \cap V_c = \emptyset$ and $E \subseteq V_m \times V_c$. (For further details on Tanner graphs, we refer the reader to [64, 69].)

Chapter 3 Junction Tree Representations for Linear Block Codes

The Generalized Distributive Law gets its efficiency by reordering the computations using the distributive law. In this chapter, we will apply the GDL to solve the decoding problem and make use of GDL's efficiency.

We will introduce a combinatorial representation for linear block codes, called the junction tree representation, which generalizes the notion of a code trellis. We will first present an algorithm for finding a minimum complexity junction tree. We will then show by examples that the minimum complexity junction tree can be less complex than the minimal trellis. One implication of this is that one can sometimes devise exact decoding algorithms which have lower complexity than those associated with the minimal trellis.

3.1 Introduction

The Generalized Distributive Law [3], which is the synthesis of the work of a long series of authors, is a general algorithm for marginalizing a "global" kernel function of many variables, whenever the kernel factors in a useful way. The key to the GDL is the notion of a junction tree [3, 28], which is a way of organizing the local domains as labels on a communications network.

In Chapter 2, we saw that decoding problems can be viewed as MPF problems with $p[U, X, Y]$ as the global kernel, where information bits $U = (u_1, \cdots, u_k)$ are encoded as code word $X = (x_1, \cdots, x_n)$ which is then transmitted and received as $Y = (y_1, \cdots, y_n)$. When $p[U, X, Y]$ factors in such a way that its local domains form a junction tree, the GDL produces exact (ML and MAP) decoding algorithms.

Trellis representations for linear block codes have a long history, and it is now a

thoroughly studied subject after the explosion of interest during the years 1995-1997. For details on this subject, we refer the reader to the chapter "Trellis Structure of Codes" in [54] and [35]. For a trellis which has $|E|$ edges and $|V|$ vertices, Viterbi's algorithm requires $|E|$ "multiplications" and $|E| - |V| + 1$ "additions." It is well known that every linear block code can be represented by a minimal trellis. McEliece [41] showed that the minimal trellis has the least value of $|E|$ among all trellises that represent a given linear block code. Later, Vardy et al. [65] proved McEliece' conjecture that the minimal trellis also minimizes $|E| - |V| + 1$. Thus, the minimal trellis requires the least number of operations when Viterbi's algorithm is applied.

It turns out that the minimal trellis associated with a code corresponds to one of the possible junction trees. When Prim's algorithm for constructing junction trees [61, 3, 28] is modified to account for complexity, we have an algorithm for finding a minimum complexity junction tree.

In this chapter, we will present some examples of codes for which the minimum complexity junction tree is less complex than the minimal trellis.

Here is an outline of the chapter. In section 2, we will analyze the complexity of the GDL when applied to decoding problems and then present an algorithm for finding a minimal complexity junction tree. In section 3, we will present some examples of codes for which the minimal complexity junction tree is less complex than the minimal trellis.

3.2 Minimum Complexity Junction Trees

In Chapter 2 we saw that the decoding problem is an instance of the MPF problem. Thus the GDL gives an efficient decoding algorithm when the global kernel factors in a useful way.

In Viterbi's algorithm, decoding begins from the initial state of the trellis and proceeds to the next state forward until it reaches the last state of the trellis. In other words, a "message" is passed from left to right in the trellis, which amounts to the "single vertex " GDL, defined in [3].

The single vertex GDL requires all but the target vertex v_i to pass one message to its neighbor. When the target vertex receives messages from all its neighbors, it updates its state, which is the required β_i. Figure 3.1 shows a junction tree of the $(8, 4, 4)$ Hamming code with generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}. \tag{3.1}$$

The arrows and the numbers above the arrows show one possible message passing schedule in the single vertex GDL. The number 1 indicates that the message is passed in the first step, and the number 2 indicates that the message is passed in the second step and so on.

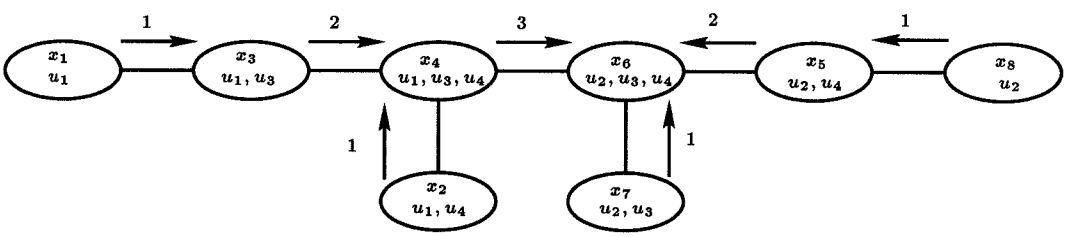


Figure 3.1: A junction tree for the $(8, 4, 4)$ Hamming code with generator matrix in (3.1). The arrows and the numbers above the arrows indicate one possible message passing schedule in the single vertex GDL.

3.2.1 The Single Vertex Generalized Distributive Law Complexity

Suppose vertex v_i (with label S_i) passes a message to vertex v_j (with label S_i). Let $d(v_i)$ be the degree of vertex v_i. Then a message update requires multiplying the $(d(v_i) - 1)$ incoming messages with the local kernel. This will take $(d(v_i) - 1)|A_{S_i}|$ multiplications in total. It is also required to marginalize out the table of $|A_{S_i}|$

elements to obtain a table of elements whose variables are also in S_j. This requires $|A_{s_i}| - |A_{S_i \cap S_j}|$ additions. To recapitulate, the message update for edge connecting v_i and v_j requires

$$(d(v_i) - 1)|A_{S_i}| \qquad \text{additions, and} \tag{3.2}$$

$$|A_{s_i}| - |A_{S_i \cap S_j}| \qquad \text{multiplications.} \tag{3.3}$$

Taking into account that $d(v_k)|A_{S_k}|$ multiplications are required for the state update at the target vertex v_k, the single vertex GDL requires

$$\sum_e \omega(e) \qquad \text{operations} \tag{3.4}$$

where $\omega(e) = |A_{S_i}| + |A_{S_j}| - |A_{S_i \cap S_j}|$, and e is the edge connecting vertices v_i and v_j.

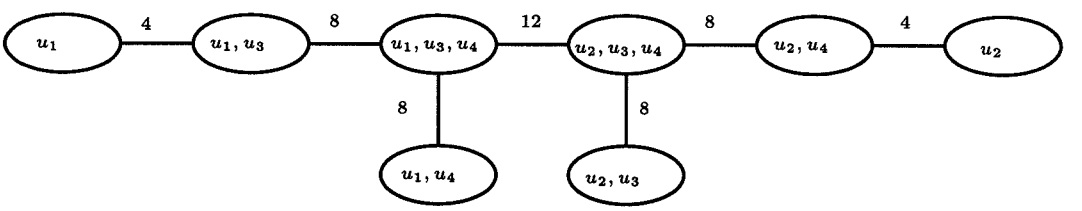


Figure 3.2: A junction tree for the $(8, 4, 4)$ Hamming code with generator matrix in (3.1). The numbers by the edges are the number of operations associated with the edges.

Figure 3.2 shows the number of operations associated with each edges. Note that the junction trees in Figure 3.2 and in Figure 3.1 represent the same $(8, 4, 4)$ Hamming code, but we do not include the x's in the vertex labels in Figure 3.2. (Messages are tables of values on u's, and x's are deterministic function of us.)

3.2.2 The Generalized Distributive Law and Trellis Complexities

The trellis complexity is the number of additions and multiplications required in Viterbi's algorithm, which takes $|E|$ multiplications and $|E| - |V| + 1$ additions [41],

where $|E|$ and $|V|$ are respectively the number of edges and vertices of the trellis.

In this complexity measure, for a binary linear code, the two edges connected to the initial state of a trellis are counted as two multiplications. These do not count in the GDL junction tree complexity measure. Similarly, the addition of the two edges connected to the finial state counts in the trellis complexity but not in GDL complexity. Thus, in this counting, for a binary linear code, the trellis complexity has 3 more operations than the GDL junction tree complexity.

3.2.3 Finding Minimum Complexity Junction Tree

Prim's algorithm as described in [13] is a *greedy* algorithm for constructing junction trees, which are maximum weight spanning trees [61, 28]. To find a minimum complexity junction tree, we have the following algorithm:

Algorithm for Finding a Minimum Complexity Junction Tree	
Step 1	Connect each pair of local domains with an edge.
Step 2	Each edge e (connecting vertices v_i and v_j) has two attributes: Weight := cardinality of $S_i \cap S_j$, Complexity := $\omega(e)$.
Step 3	Use Prim's algorithm to find a maximum weight spanning tree.
Step 4	In case of a tie in Step 3, choose edge of minimum complexity.

If this algorithm produces a junction tree, then it can be shown that the junction tree is of minimum complexity.

The junction tree in Figure 3.1 is a minimum complexity junction tree found by the above algorithm. The junction tree which corresponds to the minimal trellis of the code is shown in Figure 3.3. In this example, it turns out that both junction trees have the same complexity.

In the next section, we will show some codes whose minimum junction tree is less complex than the minimal trellis.

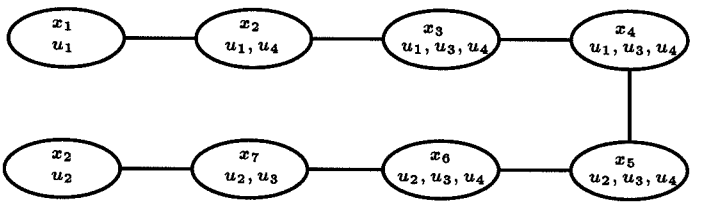


Figure 3.3: The junction tree corresponding to the minimal trellis of the $(8, 4, 4)$ Hamming code with generator matrix in (3.1).

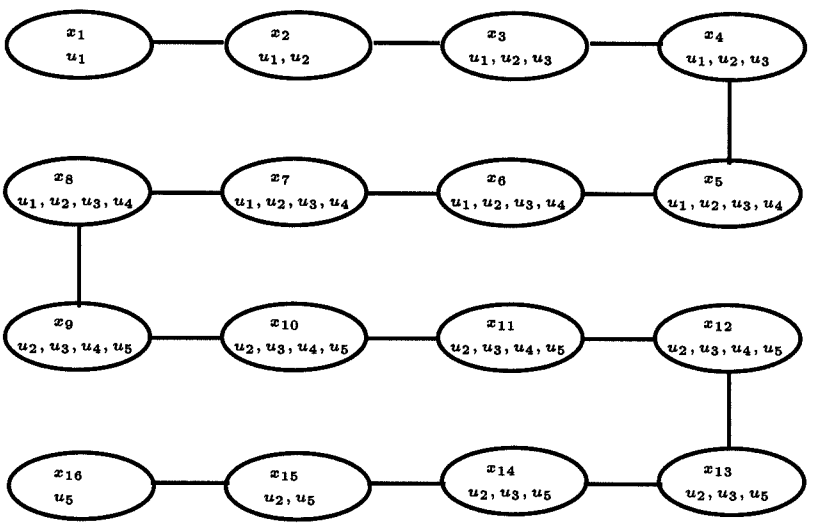


Figure 3.4: The junction tree corresponding to the minimal trellis of the $(16, 5)$ first-order RM code defined in (3.5). It requires 192 operations per decoding.

3.3 Examples

Consider the $(16, 5)$ first-order RM code defined by the generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{3.5}$$

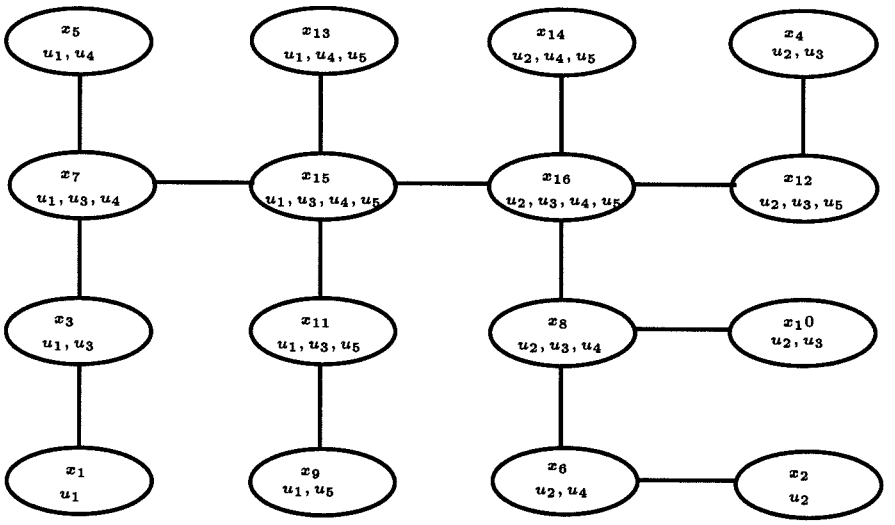


Figure 3.5: Minimum complexity junction tree corresponding to the $(16, 5)$ first-order RM code defined by G. It requires 176 operations per decoding.

A minimal span [41] generator matrix G' of G is:

$$G' = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{3.6}$$

The junction tree corresponding to the minimal trellis of this RM code is shown in Figure 3.4. Note the minimal trellis is also minimal with respect to reordering the code bits [32]. This junction tree requires 192 operations per decoding.

However, the junction tree of Figure 3.4 is not an especially good one. If we call the generator matrix obtained from G by adding row 2 and row 5 of G to row 1 of G, as $\hat{G}$, the minimal trellis complexity of $\hat{G}$ is still 192 operations because an elementary row operation does not change the minimal trellis complexity [41]. However, the minimum complexity junction tree corresponding to $\hat{G}$ shown in Figure 3.5 requires 176 operations, which is less than the minimal trellis.

In the table below, we list two other RM codes whose minimum junction tree is

less than the minimal trellis of the corresponding code. For completeness, we also include the RM $(16, 8)$ code in the table. Note that all these minimal trellises are minimal with respect to code coordinate permutations.

Code	Junction Tree Complexity (operations/decoding)	Minimal Trellis Complexity (operations/decoding)
RM$(16, 5, 8)$	176	192
RM$(32, 6, 16)$	570	731
RM$(64, 7, 32)$	1786	2827

Chapter 4 Decoding Tail-biting Trellises and Matrix Multiplication

In this chapter we will view the problem of decoding a tail-biting trellis as a problem of multiplying a chain of matrices in the min-sum semiring. The number of arithmetic operations needed to evaluate a matrix chain multiplication depends on the parenthesization; an *optimal* parenthesization is a parenthesization which requires the minimum number of operations. In the special case where the first and last matrices are a row and a column matrix respectively, we will prove a theorem on optimal parenthesizations. Using this theorem, we conclude that Viterbi's algorithm is optimal when applied to a "typical decoding trellis" in the sense that it corresponds to an optimal parenthesization. For decoding a tail-biting trellis, we are only interested in the diagonal elements of the matrix product, but computing the diagonal elements separately can require more operations than computing the full matrix product. In the case of computing the diagonal elements, the number of operations can be reduced compared to Viterbi's algorithm by finding an optimal parenthesization.

4.1 Introduction

A *tail-biting trellis* is a finite, labeled digraph in which the vertices are partitioned into n classes, $V_0, \cdots, V_{n-1}$, each class being indexed by an element of $Z_n = \{0, 1, \cdots, n-1\}$, the cyclic group of order n. Each edge goes from a vertex in V_i to a vertex in V_{i+1} for some i, with $(n-1)+1$ taken as 0 (For further details on tail-biting codes, we refer the reader to [14, 59].)

A trellis of depth n, with vertex set $V_0 \cup V_1 \ldots \cup V_n$, where $|V_0| = |V_n|$, is obtained by unwrapping a tail-biting trellis. If the weights of the edges connecting the vertices $x_{i-1} \in V_{i-1}$ and $x_i \in V_i$ are represented by a matrix $M_i[x_{i-1}, x_i]$, then the shortest

path in a trellis can be computed by matrix chain multiplication in the min-sum semiring. (The definition of min-sum semiring is given in Chapter 1.)

The Generalized Distributive Law efficiently solves a wide variety of problems, including the matrix chain multiplication problem [3]. In solving the matrix chain multiplication problem, the GDL yields an algorithm identical to Viterbi's algorithm, which corresponds to a particular parenthesization of the matrix chain.

Since the number of arithmetic operations needed to evaluate a matrix chain multiplication depends on the parenthesization, it is natural to ask if Viterbi's algorithm is optimal. This is the question we will address in this chapter.

In the rest of the chapter, we will look at a special case of the matrix chain multiplication problem corresponding to finding shortest paths in tail-biting trellises. In section 2, we will prove a theorem on optimal parenthesizations when $|V_0| = |V_n| = 1$, and we will conclude that Viterbi's algorithm is optimal when applied to a "typical decoding trellis." In section 3, we will study the $|V_0| = |V_n| > 1$ case, which corresponds to tail-biting trellises. For decoding a tail-biting trellis, we are only interested in the diagonal elements of the matrix product. There are two ways to obtain the diagonal elements. We can either compute the whole matrix product and read off the diagonal elements, or we can compute the diagonal elements separately. It may seem that computing the full matrix product requires more operations than computing the diagonal elements separately, but we will give an example in which it costs more to compute the diagonal elements of the product separately than the full matrix product.

In the remaining part of this section, we will establish the relationship between Viterbi's algorithm and matrix chain multiplication. In order to do so, we will first introduce the matrix chain multiplication problem in subsection 4.1.1. We will then cast the matrix chain multiplication problem as an instance of a MPF ("marginalizing a product function") problem [3] in subsection 4.1.2. Finally, in subsection 4.1.3, we will view Viterbi's algorithm as an algorithm for multiplying a chain of matrices in the "min-sum" semiring.

4.1.1 Matrix Chain Multiplication Problem

Let M_i be a $q_{i-1} \times q_i$ matrix for $i = 1, \ldots, n$. We denote the entries in M_i by $M_i[x_{i-1}, x_i]$, where x_i is a variable taking values in a set A_i with q_i elements, for $i = 0, 1, \ldots, n$. Suppose we wish to compute the product

$$M = M_1 \cdot M_2 \ldots M_n. \tag{4.1}$$

For $n = 2$, we have

$$M[x_0, x_2] = \sum_{x_1} M_1[x_0, x_1] M_2[x_1, x_2]. \tag{4.2}$$

For each of the $q_0 q_2$ values of $M[x_0, x_2]$ in eqn. (4.2), there are q_1 terms in the sum, each term requiring one addition and one multiplication, so that the total number of arithmetic operations required for the multiplication of 2 matrices of sizes $q_0 \times q_1$ and $q_1 \times q_2$ is $2q_0q_1q_2$.

An easy induction argument on eqn. (4.2) gives the generalization

$$M[x_0, x_n] = \sum_{x_1, \ldots, x_{n-1}} M_1[x_0, x_1] \ldots M_n[x_{n-1}, x_n], \tag{4.3}$$

for n matrices.

There are $\frac{1}{n}\binom{2n-2}{n-1}$ possible parenthesizations for the product of n matrices. Since matrix multiplication is associative, all parenthesizations yield the same product. However, the cost of evaluating the product can differ dramatically depending on the way the matrices are parenthesized.

Consider the computation of the product of three matrices. There are two different ways to parenthesize $M_1M_2M_3$. Parenthesization $M_1(M_2M_3)$ requires $2q_1q_2q_3 + 2q_0q_1q_3$ arithmetic operations, whereas parenthesization $(M_1M_2)M_3$ takes $2q_0q_1q_2 + 2q_0q_2q_3$ arithmetic operations. Thus, which parenthesization is preferred depends on the relative size of the matrices. For example, if $q_0 = 10$, $q_1 = 100$, $q_2 = 5$, and $q_3 = 50$, the first and second parenthesization require $150,000$ and $15,000$ operations, respectively.

An *optimal* parenthesization is a parenthesization which minimizes the total number of operations in evaluating the product in eqn. (4.1). Finding an optimal parenthesization is usually solved by dynamic programming [13].

4.1.2 Matrix Chain Multiplication as an Instance of the Marginalizing a Product Function Problem

A *marginalizing a product function* (MPF) problem is a problem which a "global" function of many variables factors into a product of "local" functions, and the objective of a MPF problem is to compute various marginal functions of its global function.

Computing the product, eqn. (4.3)

$$M[x_0, x_n] = \sum_{x_1, \cdots, x_{n-1}} M_1[x_0, x_1] \ldots M_n[x_{n-1}, x_n],$$

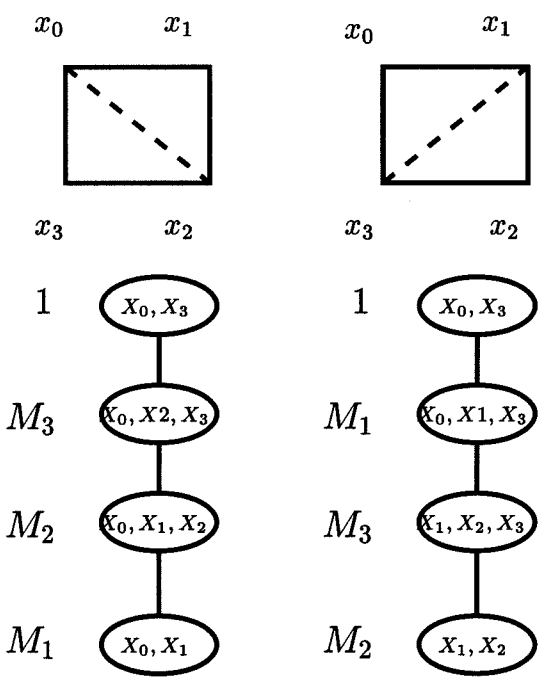


Figure 4.1: The triangulated moral graphs and the corresponding junction trees for the $n = 3$ case. The left junction tree corresponds to the parenthesization $(M_1M_2)M_3$, and the right one corresponds to $M_1(M_2M_3)$.

is an instance of the MPF problem [3]. The local domains and local kernels of the MPF problem are given in the following table.

	Local Domain	Local Kernel
1	$\{x_0, x_1\}$	$M_1[x_0, x_1]$
2	$\{x_1, x_2\}$	$M_2[x_1, x_2]$
	$\cdots$	
n	$\{x_{n-1}, x_n\}$	$M_n[x_{n-1}, x_n]$
$n+1$	$\{x_n, x_0\}$	1

If the local domains can be organized into a junction tree [28], the desired product eqn. (4.3) is the objective function at local domain $n+1$. However, the above sets of local domains do not form a junction tree for $n > 3$, because the corresponding moral graph[1] (Figure 4.2) is a cycle of length $n+1$ [28, 61, 53, 1]. The moral graph needs to be triangulated. For example, for $n = 3$ there are two different triangulations of the moral graph, as shown in Figure 4.1. The junction trees corresponding to the parenthesization $(M_1M_2)M_3$ and $M_1(M_2M_3)$ are in the lower part of Figure 4.1.

For the product of n matrices, there are $\frac{1}{n}\binom{2n-2}{n-1}$ possible triangulations of the moral graph, and they are in correspondence with the different ways of parenthesizing the n matrices. For example, the triangulation shown in Figure 4.2 corresponds to the parenthesization $((\cdots((M_1M_2)M_3)\cdots M_{n-1})M_n)$.

4.1.3 Matrix Chain Multiplication in the Min-Sum Semiring

As an alternative interpretation of eqn. (4.3), consider a trellis of depth n, with vertex set $V_0 \cup V_1 \ldots \cup V_n$. Let $M_i[x_{i-1}, x_i]$ denote the weight of the edge connecting the vertices $x_{i-1} \in V_{i-1}$ and $x_i \in V_i$. If the weight of a path is defined as the sum of the weights of the component edges, then $M[x_0, x_n]$ in eqn. (4.3) represents the sum of the weights of all paths from x_0 to x_n. For example, Figure 4.3 shows the trellis corresponding to the multiplication of three matrices of sizes 2×3, 3×3, and 3×2

[1] In a moral graph, there is an edge connecting two vertices if the two vertices belongs to a local domain. For more details on moral graph, we refer the reader to [28].

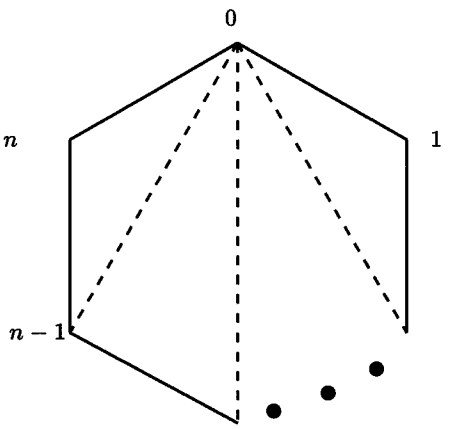


Figure 4.2: The triangulation of the moral graph corresponding to the parenthesization $((\cdots((M_1M_2)M_3)\cdots M_{n-1})M_n)$.

respectively. The (a_i, b_j) entry of the matrix product is the sum of the weights of all paths from a_i to b_j. If the computation is done in the min-sum semiring, the interpretation of eqn. (4.3) is that $M[x_0, x_n]$ is the weight of a minimum-weight path from x_0 to x_n.

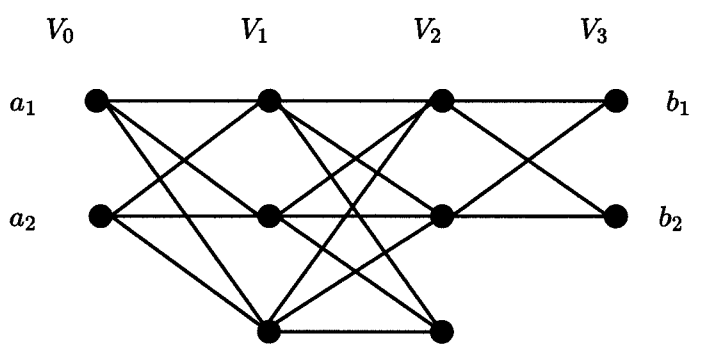


Figure 4.3: The trellis corresponding to the multiplication of three matrices, of sizes 2×3, 3×3, and 3×2 respectively. The (a_i, b_j) entry of the matrix product is the sum of the weights of all paths from a_i to b_j.

The matrix multiplication problem in the min-sum semiring is equivalent to the problem of finding the shortest paths in the trellis. If the moral graph is triangulated as shown in Figure 4.3, the resulting junction tree yields an algorithm identical to Viterbi's algorithm. Thus, Viterbi's algorithm can be viewed as an algorithm for

multiplying a chain of matrices in the min-sum semiring.

4.2 Conventional Matrix Chain Multiplication

We will call the class of matrix chain multiplication problems, where the first and last matrices have sizes $1 \times q_1$ and $q_{n-1} \times 1$, respectively, as *conventional*. (We use the term *conventional* because it corresponds to the "conventional trellis" where the initial and final vertices have a single state.)

In this section, we will show that Viterbi's algorithm corresponds to an optimal parenthesization when applied to a "typical decoding trellis," as a corollary to Theorem 4.2.2.

Denote the *left to right* and the *right to left* parenthesization of $M_1 M_2 \cdots M_n$ by $((\cdots(M_1 M_2)\cdots)M_{n-1})M_n$ and $M_1(M_2(\cdots(M_{n-1}M_n)\cdots))$ respectively. With this notation, we are ready to state Lemma 4.2.1.

Lemma 4.2.1. *For the matrix chain multiplication problem where $q_0 = 1$ and $q_i > 1$, for $i = 1, \cdots, n$, an optimal parenthesization is left to right.*

Proof: (by induction on the number of matrices n.)

1. $n = 3$.

   Let $C(M_1 \cdot M_2 \cdots M_n)$ denote the cost, i.e., the number of arithmetic operations required to perform the matrix chain multiplication.

   There are two possible ways to parenthesize three matrices, with the following corresponding costs:

$$C((M_1 M_2)M_3) = 2(q_1 q_2 + q_2 q_3), \tag{4.4}$$

$$C(M_1(M_2 M_3)) = 2(q_1 q_2 q_3 + q_1 q_3). \tag{4.5}$$

Then,

$$\frac{2(q_1q_2+q_2q_3)}{2q_1q_2q_3} = \frac{1}{q_3}+\frac{1}{q_1}, \tag{4.6}$$

$$\frac{2(q_1q_2q_3+q_1q_3)}{2q_1q_2q_3} = 1+\frac{1}{q_2}. \tag{4.7}$$

If $q_1 > 1$, then $\frac{1}{q_3}+\frac{1}{q_1} < 1+\frac{1}{q_2}$ because $q_3 > 1$, so $\mathcal{C}((M_1M_2)M_3) < \mathcal{C}(M_1(M_2M_3))$.

2. Assume the Lemma is true for $n \leq k$.

   If M_1M_2 is multiplied first in the chain of $k+1$ matrix multiplications, then by induction the proof is done.

   If M_1M_2 is not multiplied first, then suppose M_iM_{i+1} is multiplied first with $i > 1$. After the first multiplication of M_iM_{i+1}, the optimal parenthesization of $M_1 \cdots (M_iM_{i+1}) \cdots M_n$ is from left to right by the induction hypothesis. However,

$$\mathcal{C}(((\cdots(\cdots(M_1 \cdots M_{i-1})(M_iM_{i+1}))M_{i+2})\cdots M_{n-1})M_n)$$

$$> \mathcal{C}(((\cdots(\cdots(M_1 \cdots M_i)M_{i+1})M_{i+2})\cdots M_{n-1})M_n)$$

   since

$$\mathcal{C}(\cdots(M_1 \cdots M_{i-1})(M_iM_{i+1})) > \mathcal{C}((\cdots(M_1 \cdots)M_i)M_{i+1}).$$

   We have a contradiction. Thus M_1M_2 must be multiplied first. □

We have the following theorem for the conventional matrix chain multiplication problem.

Theorem 4.2.2. *For a conventional n matrix chain multiplication problem with $q_i > 1$, for $i = 1, 2, \cdots, n-1$, an optimal parenthesization is from left to right at matrix M_{i^*}, right to left at matrix M_{i^*+1}, and multiplying the two left to right and right to left matrices last (as demonstrated in Figure 4.4), where M_{i^*} is $q_{i^*-1} \times q_{i^*}$, M_{i^*+1} is $q_{i^*} \times q_{i^*+1}$, and $q_{i^*} = \arg\min_{q_i}(q_1, \cdots, q_{n-1})$.*

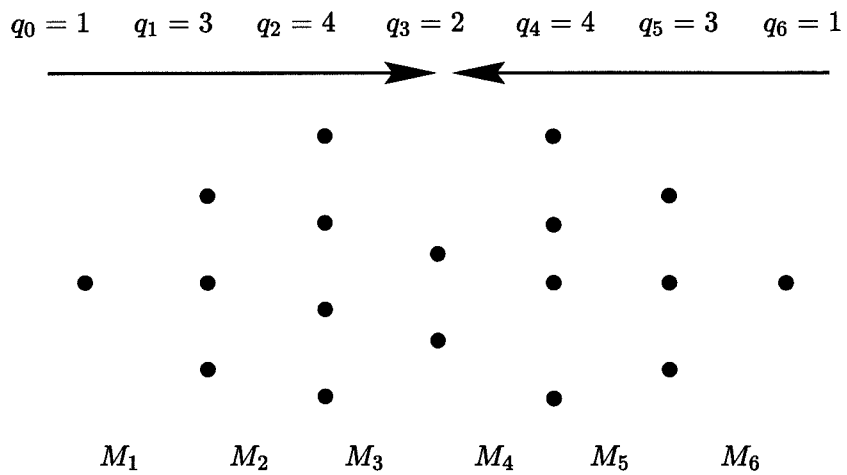


Figure 4.4: Demonstration of Theorem 4.2.2: here $M_{i^*} = M_3$, so we multiply $M_1M_2M_3$ from left to right, and $M_4M_5M_6$ from right to left. Then we multiply the last two matrices $(M_1M_2)M_3$ and $M_4(M_5M_6)$. By Theorem 4.2.2, $((M_1M_2)M_3)(M_4(M_5M_6))$ is an optimal parenthesization.

Proof: Divide the n matrices with optimal parenthesization into two groups such that the matrices which multiply M_1 before M_n fall into group A, and the rest into group B. Since it is an optimal parenthesization, A and B must be optimal as well. By Lemma 4.2.1, $A = M_1 \cdots M_i$ and $B = M_{i+1} \cdots M_n$ with $A = M_1 \cdots M_i$ ordering from left to right, and $B = M_{i+1} \cdots M_n$ ordering from right to left. Thus, the optimal parenthesization must be of the form:

$$((\cdots(((M_1M_2)M_3)\cdots))M_i)(M_{i+1}(\cdots((M_{n-2}(M_{n-1}M_n)))\cdots)).$$

Then,

$$\begin{aligned} \mathcal{C}((\cdots(((M_1M_2)M_3)\cdots))M_i) &= 2(q_1q_2 + q_2q_3 + \cdots + q_{i-1}q_i), \\ \mathcal{C}(M_{i+1}(\cdots((M_{n-2}(M_{n-1}M_n)))\cdots)) &= 2(q_iq_{i+1} + \cdots + q_{n-2}q_{n-1}). \end{aligned}$$

Since A and B are respectively $1 \times q_i$ and $q_i \times 1$ matrices, it takes $2q_i$ operations

to multiply them. Thus

$$C(((\cdots(((M_1M_2)M_3)\cdots))M_i)(M_{i+1}(\cdots((M_{n-2}(M_{n-1}M_n)))\cdots))$$
$$= 2(q_1q_2 + \cdots + q_{i-1}q_i + q_i + q_iq_{i+1} + \cdots + q_{n-2}q_{n-1}),$$

and $2(q_1q_2 + \cdots + q_{i-1}q_i + q_i + q_iq_{i+1} + \cdots + q_{n-2}q_{n-1})$ is minimum when $q_{i^*} = \arg\min_{q_i}(q_1, \cdots, q_{n-1})$. □

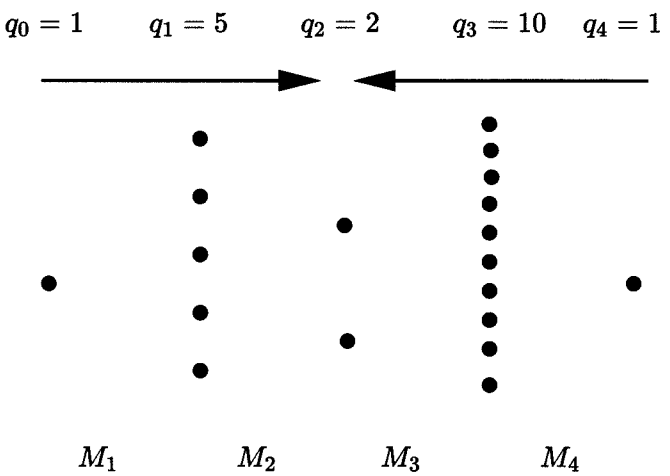


Figure 4.5: The conventional matrix chain for Example 4.2.1. $q_{i^*} = q_2$, and $((M_1M_2)(M_3M_4))$ is an optimal parenthesization.

Example 4.2.1. *Consider the conventional matrix chain multiplication problem for* $n = 4$, *with* $q_1 = 5$, $q_2 = 2$, *and* $q_3 = 10$ *(*$q_0 = q_4 = 1$*). There are five possible ways to parenthesize the four matrices. The number of arithmetic operations needed to evaluate the product of each of the parenthesizations is listed in the table below.*

Parenthesization	*Number of Operations*
$(((M_1M_2)M_3)M_4)$	80
$((M_1(M_2M_3))M_4)$	320
$((M_1M_2)(M_3M_4))$	64
$(M_1((M_2M_3)M_4))$	210
$(M_1(M_2(M_3M_4)))$	80

We see that the parenthesization $((M_1M_2)(M_3M_4))$ *requires the least number of operations, as predicted by Theorem 4.2.2. (In this case,* $q_{i^*} = q_2$ *(Figure 4.5).)*

As mentioned in the introduction, Viterbi's algorithm can be viewed as an algorithm for multiplying a chain of matrices from left-to-right in the min-sum semiring. We call a conventional trellis (with $|V_0| = |V_n| = 1$) a *typical decoding trellis* when $|V_i| \neq 1$ for $i = 1, \cdots, n-1$, and $|V_2| = |V_{n-1}| = q$, for vertex alphabet q. (A typical decoding trellis has $|V_0| = |V_n| = 1$, so $|V_2|$ and $|V_{n-1}| = q$ are necessary conditions to ensure that no two paths of a typical decoding trellis correspond the same codeword [46].) Since

$$|V_{n-1}| = \arg\min_{|V_i|}(|V_2|, \cdots, V_{n-1}),$$

by Theorem 4.2.2, one optimal parenthesization is to multiply $M_1 \cdots M_{n-1}$ from left to right. Thus, we have the following corollary.

Corollary 4.2.3. *Viterbi's algorithm corresponds to an optimal parenthesization when applied to a typical decoding trellis.*

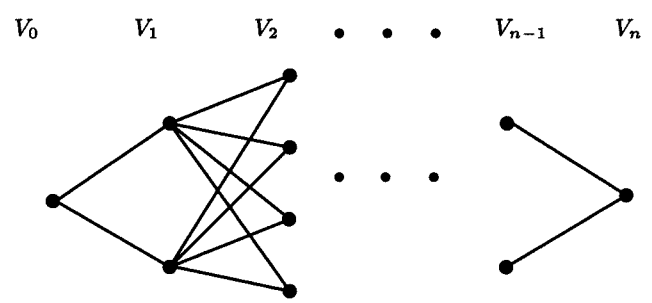


Figure 4.6: A conventional trellis with $q = 2$, where $|V_1|$ and $|V_{n-1}|$ are smallest among $\{|V_1|, \cdots, |V_{n-1}\}$. In this case, an optimal parenthesization is from left to right, as in Viterbi's algorithm.

4.3 Tail-biting Matrix Chain Multiplication

We will call the special case of matrix chain multiplication problem where the first and last matrices have size $k \times q_1$ and $q_{n-1} \times k$ respectively, the *tail-biting matrix*

chain multiplication problem. (We call it the tail-biting matrix chain multiplication problem because it corresponds to a tail-biting trellis where the initial and final vertex have the same number of states [14].)

The conventional matrix chain multiplication problem is a special case of the tail-biting matrix chain multiplication problem with $k = 1$. In this section, we are concerned with the tail-biting matrix chain multiplication problem with $k > 1$.

For decoding a tail-biting trellis, we are interested in the diagonal elements of a matrix chain multiplication product, because they correspond to the shortest tail-biting paths on the trellis. There are two ways to obtain the diagonal elements of a matrix product. We can either compute the whole matrix product and read off the diagonal elements, or we can compute the diagonal elements separately.

When the size of all matrices is the same, i.e., $q_0 = q_1 = \cdots = q_n = k$, computing only the diagonal elements requires fewer operations than computing the full matrix product. Consider the minimal tail-biting trellis for the $(24, 12, 8)$ Golay code given in [14], which has 16 states for each of the 12 sections. Computing the full matrix product requires $2(12-1)16^3 = 90,112$ operations, whereas it takes $2(12-2)16^3 + 2(16^2) = 82,432$ operations to compute the diagonal elements separately.

In general, when the size of all matrices is the same, all possible parenthesizations require the same number of operations, which is $2(n-1)k^3$. On the other hand, computing only the diagonal elements via k applications of Viterbi's algorithm requires $2(n-2)k^3 + 2k^2$ operations. Since

$$2(n-1)k^3 > 2(n-2)k^3 + 2k^2$$

for $k > 1$, it is always better to compute the diagonal elements separately.

When the size of the matrices are not all equal, using Viterbi's algorithm (we interchangeably use the terms "Viterbi's algorithm" and "multiplying a chain of matrices from left to right") to find the shortest path of a diagonal element may not be optimal. We should instead apply Theorem 4.2.2 to find an optimal parenthesization for each diagonal element. Consider the tail-biting trellis for the $(8, 4, 4)$ extended Hamming

code as given in [14]. The state or q_i profile of the trellis is $(2, 4, 4, 4, 2, 4, 4, 4, 2)$ as shown in Figure 4.7. The matrix chain product is 2×2. To compute one diagonal element of the matrix product, Viterbi's algorithm requires 168 operations, but an optimal parenthesization $(((M_1M_2)M_3)M_4)(M_5(M_6(M_7M_8)))$ requires 164 operations. (An optimal parenthesization requires 336 operations to calculate the full matrix product.)

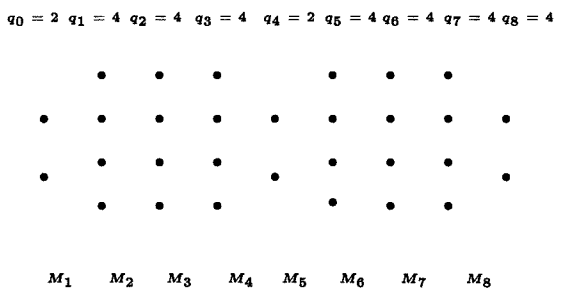


Figure 4.7: The q_i profile of the $(8, 4, 4)$ Hamming code tail-biting trellis. Viterbi's algorithm takes 168 operations per diagonal element, but an optimal parenthesization $(((M_1M_2)M_3)M_4)(M_5(M_6(M_7M_8)))$ requires 164 operations per diagonal element.

For the $(8, 4, 4)$ Hamming code tail-biting trellis, the number of operations needed to compute the full matrix product is more than the number of operations needed to compute the two diagonal elements of the product separately. However, it sometimes takes more operations to compute the diagonal elements of the product than the full matrix product, as shown in the following example.

Example 4.3.1. *Consider a tail-biting trellis shown in Figure 4.8. When we unwrap the tail-biting trellis at the* 5 *different vertices sets* V_i *for* $i = 1, \cdots, 5$, *we obtain* 5 *trellises. The state profiles of these trellises are listed in the table below. The minimum number of operations required to compute the diagonal elements separately for each trellis is listed in the table as well. We can see that the minimum number of operations needed to compute the diagonal elements separately for the tail-biting trellis is* 456.

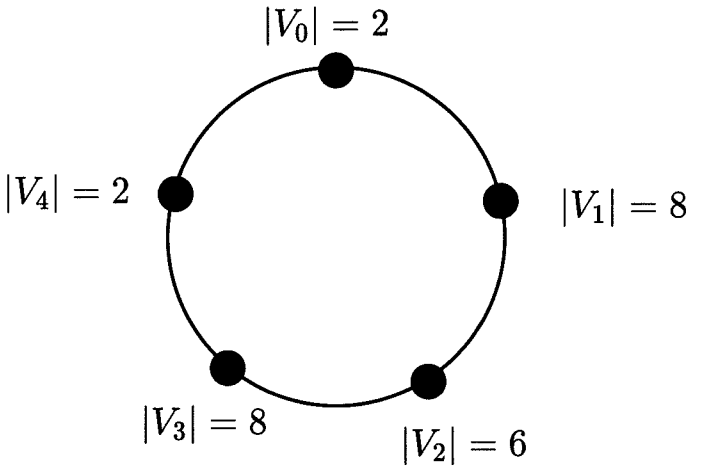


Figure 4.8: The tail-biting trellis of Example 4.3.1. While it requires 448 operations to compute the full matrix chain product, it takes 456 operations to compute the two diagonal elements separately.

State Profile	*Operations to Compute diagonal elements*
$(2, 2, 8, 6, 8, 2)$	456
$(2, 8, 6, 8, 2, 2)$	456
$(8, 6, 8, 2, 2, 8)$	1120
$(6, 8, 2, 2, 8, 6)$	456
$(8, 2, 2, 8, 6, 8)$	1120

On the other hand, computing the full matrix product requires 448 *operations with parenthesization* $M_1((M_2M_3)(M_4M_5))$ *for the first matrix chain in the table.*

4.4 Conclusion

For the special case of the matrix chain multiplication problem where the first and last matrix are a row and a column matrix respectively, Theorem 4.2.2 gives an optimal parenthesization. Using this theorem, we concluded that Viterbi's algorithm is optimal in the sense that it corresponds to an optimal parenthesization when applied to a typical decoding trellis. For decoding a tail-biting trellis, we are interested in the diagonal elements of the matrix product, but computing the diagonal elements

separately can cost more in terms of number of operations than computing the full matrix product. Even in the case of computing the diagonal elements, the number of operations can be reduced by applying Theorem 4.2.2 to find an optimal order for the corresponding conventional trellis, instead of straightforwardly using the left to right order, as in Viterbi's algorithm.

Chapter 5 Iterative Min-Sum Decoding of Tail-biting Codes

By invoking a form of the Perron-Frobenius theorem for the min-sum semiring, we obtain a union bound on the performance of iterative decoding of tail-biting codes for both the AWGN channel and the BSC. This bound shows that iterative decoding will be nearly optimum, at least for high SNRs, if and only if the minimum "pseudoweight" of the code is at least the ordinary minimum distance. We prove that the minimum pseudodistance of any 2-segment "pseudocodeword" must be at least the ordinary minimum distance. Unfortunately, the minimum pseudodistance of a 3 (or more)-segment pseudocodeword can be less than the ordinary minimum distance. We will present a tail-biting trellis for the $(24, 12, 8)$ Golay code which has a pseudoweight less than the minimum distance of the code.

5.1 Introduction

A tail-biting[1] trellis is a finite, labeled, digraph in which the vertices are partitioned into n classes $\Sigma_0, \Sigma_0, \cdots, \Sigma_{n-1}$, each class being indexed by an element of $Z_n = \{0, 1, \cdots, n-1\}$, the cyclic group of order n. (All index arithmetic is done modulo n.) If E is an edge, we denote the initial vertex of E by $\text{init}(E)$ and the final vertex of E by $\text{fin}(E)$. An edge E must have $\text{init}(E) \in \Sigma_k$ and $\text{fin}(E) \in \Sigma_{k+1}$, for some $k \in Z_n$. The label of such an edge, denoted $\text{out}(E)$ (for "output"), belongs to a finite alphabet A_k. If P is a path, the label of P, denoted $\text{out}(P)$, is the concatenation of the labels of the edges comprising P. We call $\text{out}(P)$ the output of the path P. (For further details on tail-biting trellises, we refer the reader to [14].)

[1] We have defined a tail-biting trellis in Chapter 4, and we will re-define it here again in order to use the notations for introducing a tail-biting path and a pseudocodeword.

An *L-segment tail-biting path* P is a trellis path of length Ln for which $\text{init}(P) = \text{fin}(P)$. The *code* generated by the tail-biting trellis is the set of outputs of the one-segment tail-biting paths. A *pseudocodeword* is the output of any tail-biting path of 1 or more segments.

A ML decoder would run the Viterbi's algorithm $|\Sigma_k|$ times and pick the winner among them, and an iterative min-sum decoder would start at any state and run the Viterbi's algorithm iteratively on the tail-biting trellis.

It is well known that the number of states in a tail-biting trellis may be much lower than the number of states in any ordinary trellis [59]. Wiberg et al. [70] proved a square root bound on the maximum state complexity of any tail-biting trellis for a code, which is lower bounded by the square root of the minimum possible state size of a conventional trellis at its midpoint. A tail-biting trellis is *minimal* if the square root bound is met with equality. In [14], the minimal tail-biting trellis of the $(24, 12, 8)$ Golay code and the $(8, 4, 4)$ extended Hamming code is constructed.

Because of the remarkable success of the iterative turbo-decoding algorithm [8], many coding researchers have been focusing on the study of other sub-optimal iterative decoding algorithms. Perhaps the simplest such algorithm is the iterative decoding of tail-biting codes.

We will show that iterative min-sum decoding of a tail-biting code will be effective if and only if the minimum "pseudoweight" of the code is at least its ordinary minimum weight. Closely related results (for AWGN channels) were discovered independently by Wiberg in his thesis [69] and by Forney et al. in [22].

5.2 Perron-Frobenius Theorem for the Min-Sum Semiring

In this section we will prove a "Perron-Frobenius" theorem for the min-sum semiring, which explains the behavior of the iterative decoding algorithm to be presented in section 5.3 (cf. the usual "sum-product" P.-F. theorem, e.g., [36, Theorem 4.5.12]).

Let A be an $s \times s$ irreducible matrix with entries from $\mathbb{R} \cup \{\infty\}$, with rows and columns indexed by $\{1, 2, \ldots, s\}$, and let G be the corresponding weighted digraph. Let the (i,j)th-entry of A be the weight of the edge from vertex i to vertex j. Assume that among all simple closed paths in G, there is a unique one with minimum average edge weight, and that this "critical cycle" is in fact a self-loop of weight ρ at vertex 1. (We summarize this condition by saying that A has a "simple eigenvalue.") We have the following Perron-Frobenius theorem for the min-sum semiring.

Theorem 5.2.1. *For n sufficiently large, with min-sum arithmetic,*

$$A^n = \rho^n E.$$

Here E is a fixed $s \times s$ "rank one" matrix, i.e., $E_{i,j} = x_i y_j$, where $\mathbf{x} = (x_1, \ldots, x_s)$ and $\mathbf{y} = (y_1, \ldots, y_s)$ are right and left "eigenvectors" of A with corresponding "eigenvalue" ρ.

Proof: WLOG, assume $\rho = 0$. (Otherwise, we can subtract ρ from each edge of A and call the resulting matrix A' and the weighted digraph G'.)

Let $w^*_{i,j}$ be the minimum weight path from vertex i to vertex j. Note that the (i,j)th-entry of A^n is the weight of the path $p_{i,j}$ from vertex i to vertex j with n edges. Let $W(p_{i,j})$ denote the weight of the path $p_{i,j}$.

Consider a path of length n from vertex i to vertex j. If the path contains vertex 1, we have

$$W(p_{i,j}) \geq w^*_{i,1} + w^*_{1,j}. \tag{5.1}$$

This bound is achievable if n is sufficiently large. Note that $W(p_{1,1}) = 0$.

If the path does not contain vertex 1, we can factor the path into k cycles and a simple path, for some integer k. Let a be the lowest weight of the cycles. Note that $a > 0$. Let b be the weight of the simple path. Then we have

$$W(p_{i,j}) \geq ka + b. \tag{5.2}$$

But for large k,

$$ka + b \geq w^*_{i,1} + w^*_{1,j}. \tag{5.3}$$

Thus for n sufficiently large, the least weight path of length n from vertex i to vertex j has weight $\rho n + w^*_{i,1} + w^*_{1,j}$, the (i,j)th-entry of E by $w^*_{i,1} + w^*_{1,j}$, $x_i = w^*_{i,1}$, and $y_i = w^*_{1,j}$. □

The numbers x_i and y_j have the following geometric interpretation. If we reduce each entry in A by ρ, and if we denote this modified matrix by A' and the corresponding weighted digraph by G', then x_i is the least weight of any path in G' from vertex i to vertex 1, and y_j is the least weight of any path from vertex 1 to vertex j.

5.3 Iterative Decoding of Tail-biting Codes

The iterative min-sum decoding algorithm for tail-biting codes is discussed explicitly in [59, 5, 21, 71]. Our view is that it is an application of the Generalized Distributive Law [3, 1], as applied to a junction graph with just one cycle as shown in Chapter 1. If we represent the local kernels of the single cycle junction graph with matrices, then message updates become matrix multiplication in the min-sum semiring.

In any case, if $\mathbf{y}$ is the received noisy codeword, after a finite number of iterations, the decoder will "lock on" to the pseudocodeword nearest to $\mathbf{y}$, which is called the *dominant pseudocodeword* in [21]. This follows from the min-sum Perron-Frobenius theorem (alternatively see [71] or [21]). Here the appropriate matrix A has entry $a_{i,j}$ given by $a_{i,j} = \min\{p(\mathbf{y}|\,\mathrm{out}(P) : \mathrm{init}(P) = i, \mathrm{fin}(P) = j\}$. A ML decoder will compute $\min_i\{a_{i,i}\}$, since this corresponds to the most likely tail-biting codeword. On the other hand, an iterative min-sum decoding algorithm will converge after a finite number of iterations, to the same result, *provided A has a simple eigenvalue.* In coding terms, this condition amounts to saying that there is a unique nearest pseudocodeword to $\mathbf{y}$, which is in fact a codeword. This fact allows us to bound the probability of decoder error, using the familiar union bound argument.

5.4 The Union Bound for Iterative Decoding

In this section we restrict attention to binary linear (tail-biting) codes, being used on a two-input discrete memoryless channel (DMC). A two-input DMC is shown in Figure 5.1. Note that an additive white Gaussian channel and a binary symmetric channel can be considered as special cases of a two-input DMC. A BSC is a two-input DMC with two outputs, and an AWGN channel is a two-input DMC with infinite number of outputs. We will use the insights gained in the previous section (the decoder converges to the nearest pseudocodeword) to obtain a union bound on the decoder word error probability.

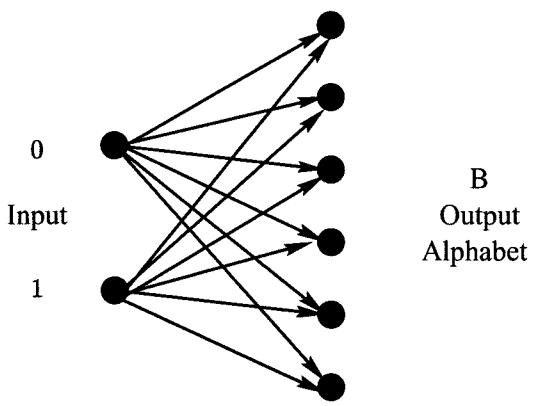


Figure 5.1: A two-input discrete memoryless channel

Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L)$ be an L-segment binary pseudocodeword, each segment $\mathbf{x}_j$ being of length n: $\mathbf{x}_j = (x_{j1}, x_{j2}, \cdots, x_{jn})$.

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{L1} & x_{L2} & \cdots & x_{Ln} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_L \end{pmatrix}$$

Let c_j (the jth column sum of $\mathbf{x}$) be defined to be

$$c_j = \sum_{i=1}^{L} x_{i,j}$$

for $j = 1, 2, \cdots, n$.

Suppose the all zero codeword $\mathbf{0}$ is transmitted over a two-input DMC with transition probabilities $p(y|0)$ and $p(y|1)$, where $y \in B$, the output alphabet. Let the received word be $\mathbf{y} = (y_1, y_2, \cdots, y_n)$. Given $\mathbf{y}$, we are interested in the probability that an iterative decoder will prefer $\mathbf{X}$ to the all zero pseudocodeword $\mathbf{0}^L = \underbrace{(\mathbf{0}, \mathbf{0}, \cdots, \mathbf{0})}_{L}$. Let $\mathbf{y}^L = \underbrace{(\mathbf{y}, \mathbf{y}, \cdots, \mathbf{y})}_{L}$ and define

$$\begin{aligned} P_1 \triangleq \Pr\{\mathbf{y}^L|\mathbf{x}\} &= \prod_{i=1}^{L} p(\mathbf{y}|\mathbf{x}_i) \\ &= \prod_{i=1}^{L}\prod_{j=1}^{n} p(y_i|x_{ij}), \\ P_0 \triangleq \Pr\{\mathbf{y}^L|\mathbf{0}\} &= \prod_{i=1}^{L}\prod_{j=1}^{n} p(y_i|0). \end{aligned}$$

Then an iterative decoder will prefer $\mathbf{x}$ to the all zero pseudocodeword $\mathbf{0}^L$ if $P_1 \geq P_0$. If we cancel the terms for which $x_{ij} = 0$, then $P_1 \geq P_0$ becomes

$$\prod_{i=1}^{n} p(y_i|1)^{c_i} \geq \prod_{i=1}^{n} p(y_i|0)^{c_i},$$

which becomes, on taking logarithms,

$$\sum_{i=1}^{n} c_i \log \frac{p(y_i|1)}{p(y_i|0)} \geq 0. \tag{5.4}$$

If we define the random variable L_i as

$$L_i \triangleq \log \frac{p(y_i|1)}{p(y_i|0)}, \tag{5.5}$$

the probability that an iterative decoder prefers $\mathbf{X}$ to $\mathbf{0}^L$ is then

$$\Pr\{\sum_{i=1}^{n} c_i L_i \geq 0\} = \Pr\{L \geq 0\}, \tag{5.6}$$

where $L \triangleq \sum_{i=1}^{n} c_i L_i$.

Example (AWGN channel)

Here B is a real line, and each y_i has the probability density function (pdf) $N(1;\sigma^2)$. If 0 is transmitted as $+1$, and 1 is transmitted as -1, we have

$$\begin{aligned} p(y|0) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-(y-1)^2}{2\sigma^2}, \\ p(y|0) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-(y+1)^2}{2\sigma^2}. \end{aligned}$$

Then

$$L_i = \log \frac{p(y_i|1)}{p(y_i|0)} = -\frac{2}{\sigma^2} y_i, \tag{5.7}$$

$$L = -\frac{2}{\sigma^2} \sum_{i=1}^{n} c_i y_i. \tag{5.8}$$

Since each y_i is $N(1;\sigma^2)$, L_i has pdf

$$N(-\frac{2}{\sigma^2}; \frac{4}{\sigma^2}). \tag{5.9}$$

Hence L has pdf

$$N(-\frac{2}{\sigma^2} \sum_{i=1}^{n} c_i; \frac{4}{\sigma^2} \sum_{i=1}^{n} {c_i}^2). \tag{5.10}$$

Thus

$$\begin{aligned}\Pr\{L \geq 0\} &= Q\left(\frac{\frac{2}{\sigma^2}\sum_{i=1}^{n} c_i}{\frac{2}{\sigma}\sqrt{\sum_{i=1}^{n} {c_i}^2}}\right), \\ &= Q\left(\frac{1}{\sigma^2}\sqrt{\frac{(\sum_{i=1}^{n} c_i)^2}{\sum_{i=1}^{n} {c_i}^2}}\right), \qquad (5.11) \\ &= Q\left(\frac{\sqrt{W(\mathbf{X})}}{\sigma}\right), \qquad (5.12)\end{aligned}$$

where $W(\mathbf{X}) \triangleq \frac{(\sum_j c_j)^2}{\sum_j {c_j}^2}$ is the AWGN pseudoweight of $\mathbf{X}$, and $Q(t) = (1/\sqrt{2\pi}) \int_t^\infty e^{-s^2/2} ds$. The AWGN pseudoweight $W(\mathbf{X})$ is called the generalized weight in [69, 22].

For example, the three-segment pseudocodeword (0000) (0101) (0011) has $c_1 = 0$, $c_2 = c_3 = 1$, and $c_4 = 2$, so that its pseudoweight is $(0+1+1+2)^2/(0^2+1^2+1^2+2^2) = 8/3$. Note that the pseudoweight of an ordinary (1-segment) codeword is the same as its usually defined weight.

Let $\mathcal{C}$ denote the set of all codewords, and $\mathcal{P}$ denote the set of all *simple* pseudocodewords (a simple pseudocodeword is one whose underlying path does not pass through the same vertex twice). The above argument implies a union bound on the (iterative min-sum) decoder word error probability for an AWGN channel $P^{\mathrm{IT}}_{E_{AWGN}}$:

$$P^{\mathrm{IT}}_{E_{AWGN}} \leq \sum_{\mathbf{x}\in\mathcal{P}} Q(\sqrt{2RW(\mathbf{X})E_b/N_0}), \qquad (5.13)$$

$$P^{\mathrm{ML}}_{E_{AWGN}} \leq \sum_{\mathbf{x}\in\mathcal{C}} Q(\sqrt{2RW(\mathbf{X})E_b/N_0}). \qquad (5.14)$$

(For comparison sake, we have also included here the ordinary union bound on $P^{\mathrm{ML}}_{E_{AWGN}}$, the maximum-likelihood word error probability.) □

Example (BSC)

Let $p\,(p < 1/2)$ be the crossover probability of the BSC.

$$P(y|0) = \begin{cases} 1-p & \text{if } y = 0 \\ p & \text{if } y = 1 \end{cases},$$

$$P(y|1) = \begin{cases} p & \text{if } y = 0 \\ 1-p & \text{if } y = 1 \end{cases}.$$

Thus

$$L_i = \begin{cases} \log \frac{p}{1-p} & \text{if } y = 0 \\ \log \frac{1-p}{p} & \text{if } y = 1 \end{cases}, \text{ and}$$

$$L = \log \frac{1-p}{p} \left\{ \sum_{i:y_i=1} c_i - \sum_{i:y_i=0} c_i \right\}. \tag{5.15}$$

Since $\log \frac{1-p}{p} > 0$, we have

$$\begin{aligned} \Pr\{L \geq 0\} &= \Pr\{ \sum_{\substack{i:y_i=1 \\ c_i \neq 0}} c_i - \sum_{\substack{i:y_i=0 \\ c_i \neq 0}} c_i \geq 0\}, & (5.16) \\ &= \Pr\{\sum_{\substack{i=1 \\ c_i \neq 0}}^{n} c_i(-1)^{y_i+1} \geq 0\}. & (5.17) \end{aligned}$$

For example, the three-segment pseudocodeword (1000) (1100) (1010) has $c_1 = 3$, $c_2 = c_3 = 1$, and $c_4 = 0$. In the table below, we list the possible $\mathbf{y}$ vectors such that $\sum_{\substack{i=1 \\ c_i \neq 0}}^{n} c_i(-1)^{y_i+1} \geq 0$, and we also include its probability $\Pr\{\sum_{\substack{i=1 \\ c_i \neq 0}}^{n} c_i(-1)^{y_i+1} \geq 0\}$.

$\mathbf{y}$	$\Pr\{\sum_{\substack{i=1 \\ c_i \neq 0}}^{n} c_i(-1)^{y_i+1} \geq 0\}$
(1000)	$p(1-p)^2$
(1100)	$p^2(1-p)$
(1010)	$p^2(1-p)$
(1110)	p^3

The error probability corresponding to this three-segment pseudocodeword is $p(1-p)^2 + 2p^2(1-p) + p^3$. Note that the error probability corresponding to an ordinary (1-segment) codeword is the error probability as usually defined. For example, the error probability of the codeword (1110) is $3p^2(1-p) + p^3$.

Define $q(\mathbf{X}, j)$ to be the number of possible $\mathbf{y}$ vectors with $|\{i : y_i = 1\}| = j$ such

that

$$\sum_{\substack{i=1 \\ c_i \neq 0}}^{n} c_i(-1)^{y_i+1} \geq 0.$$

For example, the above three-segment pseudocodeword has $q(\mathbf{X}, 1) = 1$, $q(\mathbf{X}, 2) = 2$, and $q(\mathbf{X}, 3) = 1$.

Define the BSC pseudoweight $W(\mathbf{X})$ to be

$$W(\mathbf{X}) \triangleq 2\left(\arg\min_j (q(\mathbf{X}, j) > 0)\right). \tag{5.18}$$

Thus for example, the BSC pseudoweight of the above three-segment pseudocodeword is 2.

Define $\langle \mathbf{X} \rangle$ to be the number of nonzero elements in $\mathbf{c} = (c_1, c_2, \cdots, c_n)$. Then

$$\Pr\{L \geq 0\} = \sum_{i=j}^{\langle \mathbf{X} \rangle} q(\mathbf{X}, j) p^j (1-p)^{\langle \mathbf{X} \rangle - j}. \tag{5.19}$$

Note that for an ordinary code word $\mathbf{X}$, $\langle \mathbf{X} \rangle = d$, the Hamming weight of the code word, and

$$\sum_{i=1}^{\langle \mathbf{X} \rangle} q(\mathbf{X}, i) p^i (1-p)^{\langle \mathbf{X} \rangle - j} = Q_d = \begin{cases} \sum_{r=(d+1)/2}^{d} \binom{d}{r} p^r (1-p)^{d-r} & \text{if } d \text{ is odd} \\ Q_{d-1} & \text{if } d \text{ is even.} \end{cases}$$

The above argument implies a union bound on the (iterative min-sum) decoder word error probability on a BSC $P_{E_{BSC}}^{\mathrm{IT}}$:

$$P_{E_{BSC}}^{\mathrm{IT}} \leq \sum_{\mathbf{X} \in \mathcal{P}} \sum_{j=1}^{\langle \mathbf{X} \rangle} q(\mathbf{X}, j) p^j (1-p)^{\langle \mathbf{X} \rangle - j}, \tag{5.20}$$

$$P_{E_{BSC}}^{\mathrm{ML}} \leq \sum_{\mathbf{X} \in \mathcal{C}} \sum_{j=1}^{\langle \mathbf{X} \rangle} q(\mathbf{X}, j) p^j (1-p)^{\langle \mathbf{X} \rangle - j}. \tag{5.21}$$

(Again for comparison sake, we have also included the ordinary union bound on $P_{E_{BSC}}^{\mathrm{ML}}$, the maximum-likelihood word error probability.) □

In general it is not easy to compute the pseudoweight-enumerator for a given code. However, in the next two subsections we will do so for the $(8, 4, 4)$ Hamming code.

5.4.1 The $(8, 4, 4)$ Hamming Code on an AWGN Channel

In [14, section 5.2], an optimal tail-biting trellis for the extended $(8, 4, 4)$ binary Hamming code is constructed, with state-complexity profile $(2, 4, 4, 4, 2, 4, 4, 4)$. We have used this trellis to experiment with the iterative min-sum decoding algorithm.

In Figure 5.2, we have plotted the actual performance (bit error probability) of an ML decoder and an iterative min-sum decoder (five iterations) for the $(8, 4, 4)$ Hamming code for E_b/N_0 values ranging from 0 dB to 9 dB in increments of 0.5 dB. We see no measurable difference in performance, although we know that theoretically the iterative algorithm is not as good as ML because of the presence of pseudocodewords.

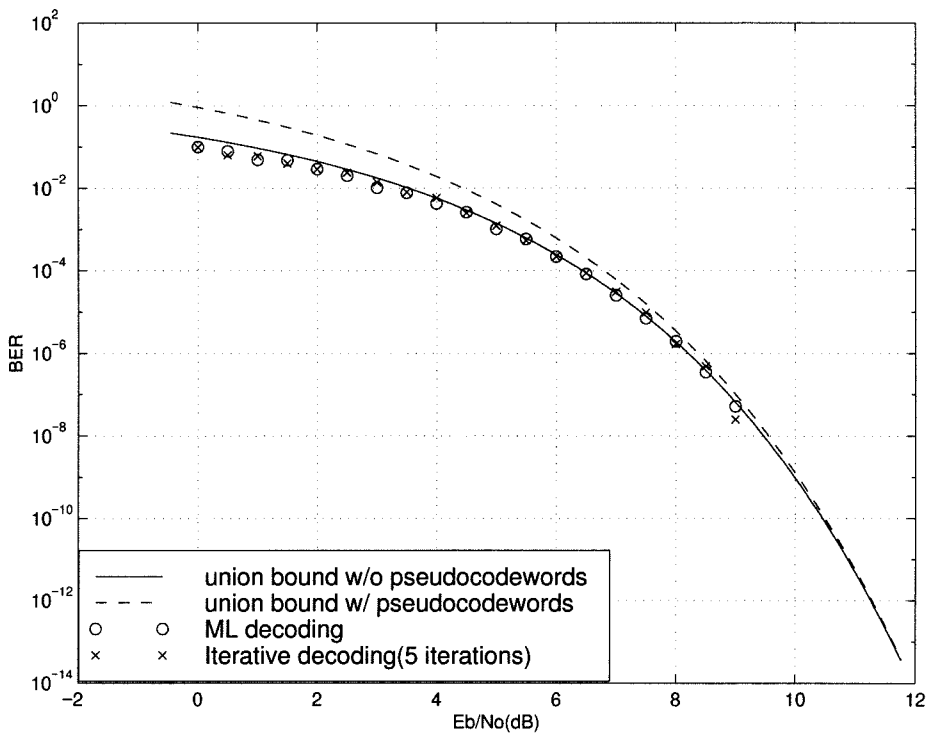


Figure 5.2: $(8, 4, 4)$ tail-biting Hamming code union bound on AWGN channel.

The pseudoweight enumerator for the $(8, 4, 4)$ Hamming code, as represented by the minimal tail-biting trellis from [14], is given in the table below. In the first row is

the ordinary weight enumerator, i.e., a list of the weights of the codewords (the one-segment pseudocodewords). In the second row is the pseudoweight enumerator for the 64 two-segment pseudocodewords. (There can be no simple pseudocodewords with more than two segments, because the trellis has state complexity 2 at two indices.)

pseudoweight	0	1	2	3	4	$4\frac{1}{2}$	5	6	$6\frac{1}{4}$	7	$7\frac{2}{17}$	8
W.E	1				14							1
P.W.E						32			30		2	

In Figure 5.2, we have plotted the bounds in equations (5.14) and (5.13), using the data from the table, modified to give bounds on bit error probability. These bounds are asymptotically equal. This is because the leading term in both bounds is the same, because the minimum pseudoweight (in this case 4.5) is strictly larger than the minimum weight (in this case 4).

5.4.2 The $(8, 4, 4)$ Hamming Code on a BSC

For the same $(8, 4, 4)$ Hamming code tail-biting trellis as in subsection 5.4.1, there are 64 pseudocodewords, and the $\mathbf{c}$'s are in one of the following three forms or are a permutation of these: $\mathbf{c}_1 = (12121211)$, $\mathbf{c}_2 = (01011210)$, and $\mathbf{c}_3 = (01211212)$. The corresponding $q(\mathbf{X}, j)$'s are given in the table below.

j	$q(\mathbf{X}, j)$ w/ $\mathbf{c}_1$	$q(\mathbf{X}, j)$ w/ $\mathbf{c}_2$	$q(\mathbf{X}, j)$ w/ $\mathbf{c}_3$
1	0	0	0
2	0	5	0
3	1	10	13
4	35	5	34
5	55	56	21
6	28	0	28
7	8	0	8
8	1	0	0

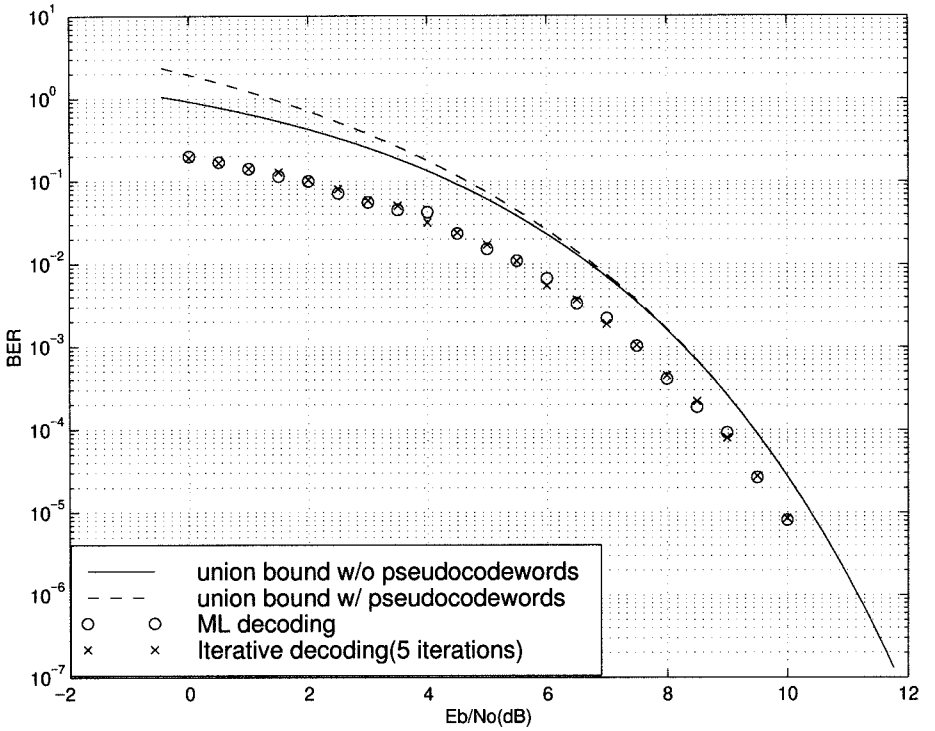


Figure 5.3: $(8, 4, 4)$ tail-biting Hamming code union bound on BSC.

In Figure 5.3, we have plotted the bounds in equations (5.20) and (5.21), using the data from the table, modified to give bounds on bit error probability. These bounds are asymptotically equal. This is because the leading term in both bounds is the same, because the minimum pseudoweight is the same as the minimum weight of the code (in both cases 4).

In Figure 5.3, we have plotted the actual performance (bit error probability) of the ML decoder and the iterative min-sum decoder (five iterations) for the $(8, 4, 4)$ Hamming code for values of E_b/N_0 ranging from 0 dB to 10 dB in increments of 0.5 dB. We see no measurable difference in performance.

5.5 Pseudoweights on an AWGN Channel

Because of the union bound in eqn. (5.13), if the minimum pseudoweight is at least the minimum weight for a given code, then the performance of iterative min-sum decoding will be asymptotically the same as ML decoding. The problem remains to find the

pseudoweights. The following lemma shows that any 2-segment pseudocodeword has a pseudoweight at least equal to the minimum weight when the 2 segments are not identical.

Lemma 5.5.1. *Let* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *be n-dimensional vectors in Euclidean space. Then*

$$\frac{|\mathbf{x}_1|^2 + |\mathbf{x}_2|^2}{|\mathbf{x}_1 + \mathbf{x}_2|} \geq |\mathbf{x}_1 - \mathbf{x}_2|.$$

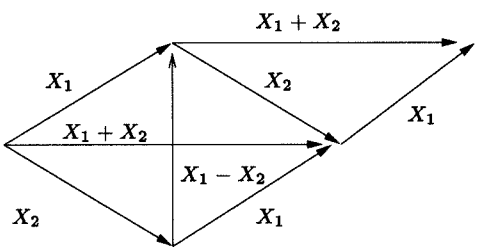


Figure 5.4: 2-segment pseudocodeword in Euclidean space.

Proof:

$$\begin{aligned} 2|\mathbf{x}_1 - \mathbf{x}_2||\mathbf{x}_1 + \mathbf{x}_2| &\leq |\mathbf{x}_1 - \mathbf{x}_2|^2 + |\mathbf{x}_1 + \mathbf{x}_2|^2, \\ &= 2|\mathbf{x}_1|^2 + 2|\mathbf{x}_2|^2, \end{aligned}$$

(See Figure 5.4 for the last equality.) □

Theorem 5.5.2. *The pseudoweight of any 2-segment nonzero pseudocodeword must be at least the minimum weight if the 2 segments are not identical.*

Proof: From the definition of pseudoweight, a 2-segment $w(\mathbf{x})$ can be expressed as:

$$\begin{aligned} \sqrt{w(\mathbf{x})} &= \frac{|\mathbf{x}_1|^2 + |\mathbf{x}_2|^2}{|\mathbf{x}_1 + \mathbf{x}_2|} \\ &\geq |\mathbf{x}_1 - \mathbf{x}_2|. \end{aligned}$$

(the last inequality is by the Lemma 5.5.1). But $\mathbf{x}_1 - \mathbf{x}_2$ is an ordinary (tail-biting) codeword, and $|\mathbf{x}_1 - \mathbf{x}_2|^2$ is the codeword weight. □

If we assume the tail-biting trellis is "n-observable," i.e., P can be inferred from out(P) for paths of length $\geq n$, then even if $\mathbf{x}_1 = \mathbf{x}_2$ the conclusion of Theorem 5.5.2 holds.

Because of the union bound, for tail-biting trellises where the minimum number of states is 2, by Theorem 5.5.2, the performance of iterative min-sum decoding will be asymptotically the same as ML decoding. For tail-biting trellises where the minimum number of states is greater than 2, the 2-segment pseudocodewords are not all the possible pseudocodewords. However, the pseudoweight of a 3 (or more)-segment pseudocodeword can be less than the minimum distance. Kötter and Vardy have constructed a class of tail-biting trellises whose minimum pseudoweight is strictly less than the minimum distance of the code for a 3 (or more)-segment pseudocodeword [31].

The question of finding the minimum pseudoweight remains. Unfortunatly, the number of pseudocodewords grows exponentially with the product of the length of the code and the number of segments. Finding the minimum pseudoweight by searching over all possible pseudocodewords is infeasible. We use a branch and bound search to find the minimum pseudoweight. The bound in the branch and bound search is presented in the next section. In the next example, we use the branch and bound search to find the minimum pseudoweight for two tail-biting trellises of the $(24, 12, 8)$ Golay code, and we find a 4-segment pseudocodeword whose pseudoweight is 7.36 for one of the tail-biting trellises.

Example 5.5.1. *For the* $(24, 12, 8)$ *Golay code, a* 16*-state minimal tail-biting trellis is constructed by Calderbank et al. [14], and there also is a* 64*-state time invariant tail-biting trellis of the code with generator* $(1 + D^5 + D^6, 1 + D + D^2 + D^4 + D^5)$ *[6]. The encoder for the* 64*-state tail-biting trellis representation of the code is shown in Figure 5.5.*

For the 64*-state tail-biting trellis, using branch and bound, we find a* 4*-segment pseudocodeword with pseudocodeword* 7.36*, which is less than* 8*, the minimum distance*

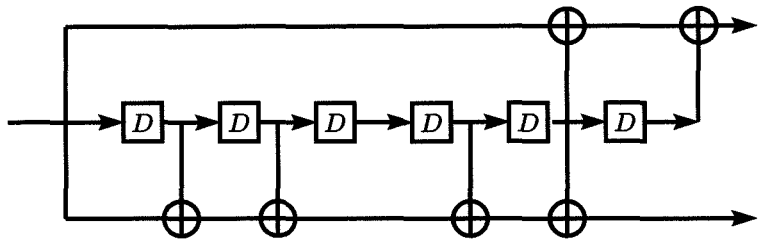


Figure 5.5: The encoder for the 64-state tail-biting trellis of the $(24, 12, 8)$ Golay code.

of the code. The 4*-segment pseudocodeword is*

00	00	00	00	00	00	00	00	00	00	11	10
00	10	00	00	00	01	01	00	00	00	10	10
10	00	00	00	00	01	00	00	00	00	01	10
01	01	00	00	00	01	00	01	00	00	01	10

For the 16*-state tail-biting trellis, we are able to find the minimum pseudoweights for up to* 4*-segment pseudocodewords. In the table below, we list the minimum pseudoweights of the* 1*-segment to* 4*-segment pseudocodewords.*

Number of Segments	*Minimum pseudoweight*
1	8
2	8
3	8
4	9.09
≥ 5	?

The minimum pseudoweights of 5*(or more)-segment pseudocodewords are still unknown, and we conjecture that they are at least the minimum distance of the code.*

We have plotted the iterative min-sum decoding performance of both trellises as well as the ML decoding performance in Figure 5.6. While there is not much difference between the performance of ML decoding and iterative min-sum decoding of the 16*-state tail-biting trellis, there is about* 0.5 *dB difference between ML decoding and iterative min-sum decoding of the* 64*-state tail-biting trellis, due to the low weight*

pseudocodeword.

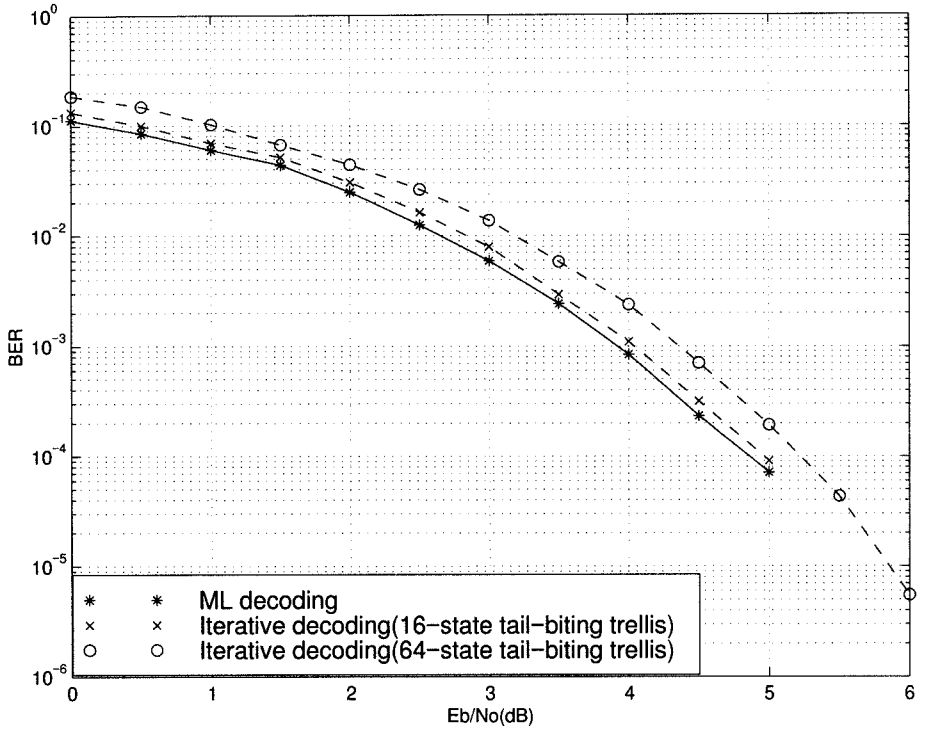


Figure 5.6: $(24, 12, 8)$ tail-biting Golay code performance on AWGN channel.

5.5.1 Branch and Bound Algorithm

Since the number of pseudocodewords grows exponentially with the product of the length of the code and the number of segments, searching over all possible pseudocodewords to find the minimum pseudoweight is infeasible. We present a branch and bound search here for finding the minimum pseudoweight.

From the definition of pseudoweight, an L-segment $w(\mathbf{x})$ can be expressed as

$$\begin{aligned} w(\mathbf{x}) &= \frac{(\sum_j c_j)^2}{\sum_j c_j^2}, \\ &= \left(\frac{\sum_{i=1}^{L} |\mathbf{x}_i|^2}{|\sum_{i=1}^{L} \mathbf{x}_i|} \right)^2 . \end{aligned} \tag{5.22}$$

For example the two-segment pseudocodeword (1101) (0110) has $c_1 = 1$, $c_2 = 2$, $c_3 = c_4 = 1$, $|\mathbf{x}_1|^2 = 3$, $|\mathbf{x}_2|^2 = 2$, and $|\mathbf{x}_1 + \mathbf{x}_2| = \sqrt{1^2 + 2^2 + 1^2 + 1^2} = \sqrt{7}$, Its

pseudoweight is therefore

$$\frac{(1+2+1+1)^2}{1^2+2^2+1^2+1^2} = \left(\frac{3+2}{\sqrt{7}}\right)^2.$$

Further, we have

$$\begin{aligned}
\frac{\sum_{i=1}^{L}|\mathbf{x}_i|^2}{|\sum_{i=1}^{L}\mathbf{x}_i|} &\geq \frac{\sum_{i=1}^{L}|\mathbf{x}_i|^2}{\sum_{i=1}^{L}|\mathbf{x}_i|}, \\
&= \frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2+\sum_{i=k+1}^{L}|\mathbf{x}_i|^2}{\sum_{i=1}^{k}|\mathbf{x}_i|+\sum_{i=k+1}^{L}|\mathbf{x}_i|}, \\
&\geq 2\left(\sqrt{\frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2}{(L-k)^2}+\frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2}{L-k}}-\frac{\sum_{i=1}^{k}|\mathbf{x}_i|}{L-k}\right), \qquad (5.23)
\end{aligned}$$

where the last inequality follows from the simple calculus fact that

$$2\left(\sqrt{\frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2}{(L-k)^2}+\frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2}{L-k}}-\frac{\sum_{i=1}^{k}|\mathbf{x}_i|}{L-k}\right)$$

is the minimum of

$$\frac{\sum_{i=1}^{k}|\mathbf{x}_i|^2+\sum_{i=k+1}^{L}|\mathbf{x}_i|^2}{\sum_{i=1}^{k}|\mathbf{x}_i|+\sum_{i=k+1}^{L}|\mathbf{x}_i|}.$$

In view of eqn. (5.23), the pseudoweight of an L-segment pseudocodeword can be bounded by knowing the first k-segment of the pseudocodeword, for $k = 1, 2, \cdots, L-1$. We then have the following branch and bound searching algorithm.

The branch and bound algorithm proceeds as the follows. For $k = 1, \cdots, L-1$, we find all the k-segment pseudocodewords, and we use eqn. (5.23) to bound their corresponding L-segment pseudoweights. We disregard those pseudocodeword branches whose pseudoweights are larger than a preset weight. Otherwise, we keep the remaining low weight pseudocodeword "branches" and repeat the process with a $k = k+1$ on the remaining branches. At $k = L-1$, if the branch and bound algorithm does not find a pseudocodeword whose pseudoweight is lower than the preset weight, then the minimum pseudoweight of the tail-biting trellis is at least the preset weight.

The bound in eqn. (5.23) is weak, so this branch and bound algorithm is compu-

tationally infeasible for larger L.

5.6 Pseudoweights on the BSC

If the BSC pseudoweight is at least as large as the minimum Hamming weight of the code, then the iterative min-sum decoder performs asymptotically as well as a ML decoder. For a 2-segment pseudocodeword we have the following theorem for its pseudoweight.

Theorem 5.6.1. *The BSC pseudoweight of any nonzero 2-segment pseudocodeword must be at least the minimum weight if the 2 segments are not identical.*

Proof: WLOG, rearrange the pseudocodeword fragments $\mathbf{x}_1$ and $\mathbf{x}_2$ so that

$$\begin{aligned}
\mathbf{x}_1 &= 1\ldots1\ 0\ldots0\ 1\ldots1\ 0\ldots0, \\
\mathbf{x}_2 &= 1\ldots1\ 1\ldots1\ 0\ldots0\ 0\ldots0, \\
\mathbf{x}_1 \oplus \mathbf{x}_2 &= \underbrace{0\ldots0}_{n_1}\ \underbrace{1\ldots1\ 1\ldots1}_{n_2}\ 0\ldots0, \\
\mathbf{c} &= \underbrace{2\ldots2}_{n_1}\ \underbrace{1\ldots1\ 1\ldots1}_{n_2}\ 0\ldots0,
\end{aligned}$$

where $\mathbf{c}$ is the column sum (ordinary sum) of the pseudocodeword fragments. Let d_{min} denote the ordinary minimum distance. Since $\mathbf{x}_1 \oplus \mathbf{x}_2$ is an ordinary codeword, $n_2 \geq d_{min}$. Hence, it suffices to show $W(\mathbf{X}) \geq n_2$. There are two cases to be considered.

Case 1: $n_1 \geq \lceil \frac{n_2}{2} \rceil$.

$$\begin{aligned}
W(\mathbf{X}) &\geq \frac{1}{2}(2n_1 + n_2), \\
&\geq \lceil \frac{n_2}{2} \rceil + \frac{n_2}{2}.
\end{aligned}$$

Case 2: $n_1 < \lceil \frac{n_2}{2} \rceil$. Let integer k be $k = \frac{W(\mathbf{X})}{2} - n_1$. Then,

$$\begin{aligned} 2n_1 + (\frac{W(\mathbf{X})}{2} - n_1) &\geq \frac{1}{2}(2n_1 + n_2), \\ \frac{W(\mathbf{X})}{2} &\geq \frac{n_2}{2}. \end{aligned}$$

Again, the BSC pseudoweight of a 3 (or more)-segment pseudocodeword can be less than the minimum distance of a code. In particular, the 4-segment pseudocodeword of the 64-state tail-biting trellis for the $(24, 12, 8)$ Golay code in section 5.5 has a pseudoweight of 6. For the 16-state minimal tail-biting trellis of the Golay code, there are 3-segment and 4-segment pseudocodewords whose minimum BSC pseudoweights are less than 8. For example, the following 3-segment pseudocodeword has BSC pseudoweight 6.

00	00	00	00	00	00	00	00	00	00	00	11
01	00	01	00	00	01	00	01	00	01	00	11
00	00	01	10	10	00	00	01	00	00	01	11

5.7 Conclusions

Because of the union bound, if the minimum pseudoweight is at least the minimum weight, then the performance of iterative min-sum decoding will be asymptotically the same as ML decoding. For a tail-biting trellis with a minimum state complexity of two, its pseudoweight will not be less than the minimum weight of the code. However, this is not true for a tail-biting trellis whose state complexity is three or more, and its minimum pseudoweight may depend on the structure of the tail-biting trellis. For the $(24, 12, 8)$ Golay code, while the 64-state tail-biting trellis has a low weight pseudocodeword, we conjecture that the minimum AWGN pseudoweight of the 16-state minimal tail-biting trellis is at least the minimum distance. The bound in our branch and bound algorithm is weak, so the branch and bound algorithm is computationally infeasible for finding minimum pseudoweight of a tail-biting trellis whose minimum state complexity is large. It remains a challenging problem to come

up with a practical algorithm for computing the minimum pseudoweight.

Chapter 6 Iterative Min-Sum Decoding of Cycle Codes on BSCs

Cycle codes are LDPC codes in which each codeword bit is checked by exactly two parity checks. In this chapter we will study the behavior of iterative decoding on Tanner graphs for cycle codes. We will present a union bound for an iterative min-sum decoder word error probability over a binary symmetric channel, and we will show that an iterative min-sum decoder performs asymptotically as well as a ML decoder at high SNRs using the union bound argument.

6.1 Introduction

Cycle codes were introduced by Gallager [25] as a special type of low density parity check (LDPC) codes in which each codeword bit is checked by exactly two parity checks. He proposed a simple hard decision iterative decoding scheme on a BSC. In his decoding scheme, the decoder first makes a decision on each codeword bit, then computes the parity checks and changes any codeword bit that is contained in more than a fixed number of unsatisfied parity checks. The process is repeated using the changed codeword bits. For a k-regular cycle code[1], Gallager showed that the error probability at the $i+1$ iteration p_{i+1} obeys the recursion $p_{i+1} = p - p\left[\frac{1+(1-2p_i)^{k-1}}{2}\right] + (1-p)\left[\frac{1+(1-2p_i)^{k-1}}{2}\right]$, where p is the crossover probability of the BSC. It is easy to see that $p_i \to \frac{1}{2}$ as $i \to \infty$. (This result can be easily extended to an irregular cycle code[2].) Note that for cycle codes, this decoding scheme is worse than no coding.

[1] A k-regular cycle code is a cycle code where each check checks k codeword bits.

[2] Gallager's analysis applied to the family of regular LDPC codes whose graphs have infinite girth. Luby et al. [38] extended Gallager's work to irregular LDPC codes whose underlying graphs have short cycles. Richardson and Urbanke [57] extended the analysis further to include belief propagation decoding.

Gallager also proposed a belief propagation decoding (iterative sum-product type decoding) scheme. He stated that this decoding scheme "decodes with a lower error probability," but studies have not been done on what the lower error probability is for cycle codes. Recently, Wiberg analyzed the iterative min-sum decoding of cycle codes and proved a necessary condition for a decoding error [69, 70].

We will extend Wiberg's analysis and show that the iterative min-sum decoder is nearly as good as a ML decoder by using a union bound argument.

6.2 Message Passing on Tanner Graphs

A cycle code is a LDPC code where there are exactly two ones in each column (the number of ones in each row is not fixed) of its parity check matrix. For example, a $(7, 3, 3)$ cycle code has the following parity check matrix H:

$$H = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}. \tag{6.1}$$

Note that the last row in H is the sum of all the other rows. The parity matrix has redundancy in it.

We have labeled row i of H by c_i and column j of H by v_j. If we consider H as a incidence matrix where a 1 in the (i, j)th entry of H indicates a connection from vertex c_i to vertex v_j, we obtain the so called Tanner graph as shown in Figure 6.1. (The Tanner graph can also be obtained as a special type of junction graph as shown in Chapter 1.)

Each vertex v_i corresponds to a code bit, so we call it a codeword node. Also we call a vertex c_j a check node in a Tanner graph.

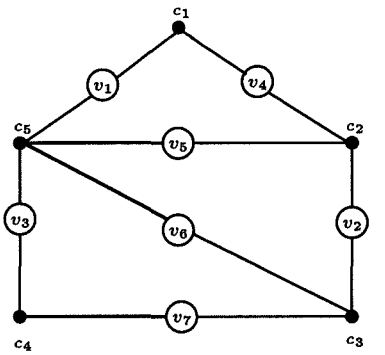


Figure 6.1: Tanner graph of the $(7, 3, 3)$ cycle code whose parity matrix is given in eqn. (6.1). Codeword nodes and check nodes are labeled by $v_i, i \in [n]$, and c_j respectively.

We will analyze the iterative min-sum message passing on the Tanner graph of a cycle code. It is well known that for cycle free graphs, message passing results in a ML or a MAP decision [42, 3, 23, 53]. However, message passing is known to be suboptimal for graphs containing cycles, such as the Tanner graph of a cycle code. Nevertheless, there is a large body of empirical evidence illustrating the excellent performance of message passing algorithms on graphs with cycles [58, 50]. In fact, we will show that an iterative min-sum decoder performs asymptotically as well as a ML decoder at high SNRs.

6.2.1 Message Passing Algorithms

The channel model we consider is the memoryless binary symmetric channel. The binary codeword vector $\mathbf{x} = (x_1, \cdots, x_n)$ is transmitted across the channel and received as a vector $\mathbf{y} = (y_1, \cdots, y_n)$. Each x_i corresponds to a codeword node v_i in the Tanner graph.

The class of message passing algorithms which we consider is described as follows. Each codeword node $v_i, i \in [n]$, has an associated channel output y_i. One iteration of message passing consists of the following two steps. First, each codeword node sends a message to each neighboring check node c_j. Then, every check node c_j sends a message to each neighboring codeword node v_i.

The message from codeword node v_i to check node c_j along the edge $e := (v_i, c_j)$ depends on the received y_i and all the incoming messages from the edges connected to v_i except the one along edge e. The message from a check node c_j to a codeword node v_i along the edge $e := (c_j, v_i)$ depends on all the incoming messages from the edges connected to c_j except the one along edge e. The requirement of excluding the incoming message along the edge e is an important property of good message passing algorithms; in turbo coding terminology, it says that only the *extrinsic* information is passed. The detailed message updating rule is described in Chapter 1, in which we view it as an instance of the Generalized Distributive Law.

After some iterations, the decision (0 or 1) of each node in the Tanner graph is based on the state of the node, which depends on all the messages incident on the node and the channel received value associated with the node if it is a codeword node.

6.2.2 Computation Trees and Deviations

By examining the message updates that have occurred after a fixed number of iterations, we can trace back recursively the computation of the messages to a codeword node v_i and all the messages which in term depend on these messages. This trace back has the form of a tree which consists of the codeword nodes and check nodes interconnected in the same way as the original Tanner graph. Such a tree is called a computation tree for codeword node v_i, and the codeword node v_i is the root of the tree.

Figure 6.2 shows a computation tree for codeword node v_1 for the $(7, 3, 3)$ cycle code of Figure 6.1 after 2 iterations of message passing. When walking down the tree towards the leaves[3], we will encounter check nodes and codeword nodes in the previous iteration until we are at the leaves where the message passing begins. In general, codeword nodes and check nodes appear many times in the computation tree. For example, codeword node v_2 occurs three times in Figure 6.2, and this implies that the information from v_2 would be counted three times if a decision for v_1 were made

[3]The leaves are always codeword nodes.

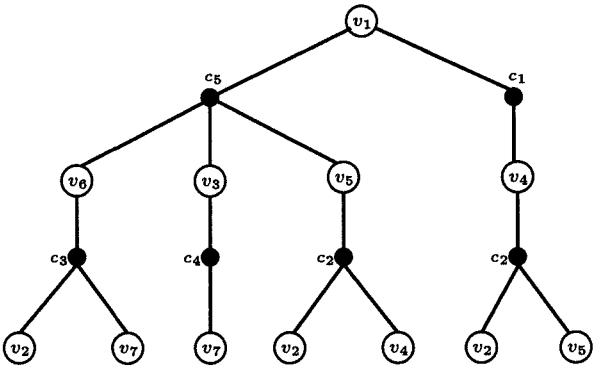


Figure 6.2: The computation tree for the $(7, 3, 3)$ cycle code of Figure 6.1 after 2 iteration of message passing.

after 2 iterations.

Assume the all zero codeword is transmitted, Wiberg [69] defined a *deviation* as a locally consistent set of codeword nodes and check nodes in the computation tree with the root node being one. Since each codeword node is checked by two check nodes, the deviation of a cycle code is a walk starting from a leaf codeword node, going through the root node, and ending in another leaf codeword node, with all the codeword nodes having the value one.

Deviations play the role of error events in the iterative min-sum decoding. Wiberg [69] analyzed the deviations and showed that a necessary condition for an iterative min-sum decoding error to occur after sufficiently many decoding iterations is that there should be a *non-cycle irreducible closed walk*. A non-cycle irreducible closed walk is defined in the following section.

6.3 Iterative Min-Sum Decoding of Cycle Codes

In this section, we will prove that the pseudoweight of a *non-cycle irreducible closed walk* (NCICW) is at least as large as the minimum distance of the code. Using a union bound argument, we will show that an iterative min-sum decoder performs as well as a ML decoder at high SRNs.

First, we will use the definitions from Wiberg [69] to define a NCICW.

Definition. *A* walk *is a sequence of codeword nodes* $v_1, v_2, \cdots$ *corresponding to a connected sequence of alternating codeword nodes and check nodes on the Tanner graph such that same edge is never utilized twice in a row*[4].

Definition. *A* closed walk *is a finite walk* $v_1 \cdots v_n$ *such that the concatenating of the walk with itself, i.e.,* $v_1, \cdots, v_n, v_1, \cdots, v_n$, *is also a walk.*

Definition. *An* reducible walk $v_1, v_2, \cdots$ *is a walk such that, for some* i *and some* $j > i$, *the segment* $v_i, \cdots, v_j$ *is a closed walk and the remaining part* $v_1, \cdots, v_{i-1}, v_{j+1}, \cdots$, *is still a walk; An* irreducible walk *is a walk that is not reducible.*

Definition. *A* cycle *is an irreducible closed walk in which every codeword node is distinct; a* NCICW *is an irreducible closed walk which is not a cycle.*

Note that cycles and disjoint union of cycles are the codewords of a cycle code. If all the possible errors were the errors made by cycles, then an iterative min-sum decoder would perform as well as a ML decoder. However, decoder error can also made by NCICWs, and NCICWs play the same role as pseudocodewords in the iterative min-sum decoding of tail-biting codes. We will examine the pseudoweight of a NCICW in the following subsection.

6.3.1 Pseudoweight of a NCICW

Before we will examine the pseudoweight of a NCICW, we will need the following lemmas.

Lemma 6.3.1. Wiberg [69, Lemma 6.3] *No codeword node occurs more than twice in an irreducible closed walk.*

Lemma 6.3.2. Wiberg [69, Lemma 6.4] *Any NCICW contains at least two cycles which may be partially overlapping, or disjoint.*

[4]This requirement follows from the updating rule of message passing where the incoming message from the same edge is excluded from the computation of the outgoing message of the edge.

Let $\mathbf{p} = (p_1, \cdots, p_L)$ be a NCICW of length L, and let c_i be the number of times a codeword node v_i appears in $\mathbf{p}$ for $i \in [n]$. Let the definition of pseudoweight be the same as that of a pseudocodeword for tail-biting codes. Define $q(\mathbf{p}, j)$ to be the number of ways of choosing j of $y_i = 1$ such that

$$\sum_{\substack{i:y_i=1 \\ c_i \neq 0}} c_i - \sum_{\substack{i:y_i=0 \\ c_i \neq 0}} c_i \geq 0. \tag{6.2}$$

Define

$$\hat{\imath} := \arg \min_i (q(\mathbf{p}, i) \neq 0). \tag{6.3}$$

Define the pseudoweight of a NCICW $w(\mathbf{p})$ to be

$$w(\mathbf{p}) = 2\hat{\imath}. \tag{6.4}$$

Let d be the minimum distance of a cycle code. Let a be the number of codeword nodes which occur once and let b be the number of codeword nodes which occur twice in the NCICW $\mathbf{p}$.

Lemma 6.3.3. *$a \geq d$.*

Proof: By Lemma 6.3.1, if the two cycles in a NCICW are disjoint, then $a \geq 2d$ because d is the length of the shortest cycles. Now suppose the two cycles in a NCICW are partially overlapping. It is impossible for the two cycles to share $\geq \frac{d}{2}$ vertices because otherwise a cycle of length $< d$ can be constructed using the unshared vertices. Thus, the two cycles can share only $\leq \frac{d}{2}$ vertices, and $a \geq d$.
□

Lemma 6.3.4. *Every NCICW $\mathbf{p}$ has $w(\mathbf{p}) \geq a$.*

Proof: There are two cases to be considered.

Case 1: $b \geq \lceil \frac{a}{2} \rceil$.

$$\begin{aligned} w(\mathbf{p}) &\geq \frac{1}{2}(a+2b), \\ &\geq \frac{a}{2} + \lceil \frac{a}{2} \rceil. \end{aligned}$$

Case 2: $b < \lceil \frac{a}{2} \rceil$.

$$\begin{aligned} 2b + (\frac{w(\mathbf{p})}{2} - b) &\geq \frac{1}{2}(a+2b), \\ \frac{w(\mathbf{p})}{2} &\geq \frac{a}{2}. \end{aligned}$$ □

Theorem 6.3.5. *Every NCICW* $\mathbf{p}$ *has* $w(\mathbf{p}) \geq d$.

Proof: By Lemma 6.3.4 and Lemma 6.3.3. □

Definition. *A* bad *NCICW is a NCICW which contains two disjoint closed walks connected by a path on which each node is visited twice.*

In [27], Horn showed that a necessary condition for an iterative min-sum word error to occur is that there has to be bad NCICWs, and he showed that the pseudoweight of a bad NCICW is at least $2d$ on an AWGN channel. We have the following theorem for the pseudoweight of a bad NCICW on a BSC.

Theorem 6.3.6. *Every bad NCICW* $\mathbf{p}$ *has* $w(\mathbf{p}) \geq 2d$ *on a BSC.*

Proof: By Lemma 6.3.4, $w(\mathbf{p}) \geq a$, so it suffices to show $a \geq 2d$. If the two closed walks in $\mathbf{p}$ are cycles, then $a \geq 2d$. If one of the closed walk is a NCICW, let the number of codeword nodes which occur once in this closed walk be a_1. Then by Lemma 6.3.3, $a' \geq d$. Thus, $a \geq (a_1 + d) \geq 2d$. Similarly, if both the closed walks are NCICW, let a_1, a_2 be the number of codeword nodes which occur once in the two NCICWs, respectively. Then $a \geq a_1 + a_2 \geq 2d$. □

6.3.2 The Union Bound for Iterative Min-Sum Decoding

In this section we will state without proof [5] a union bound on the performance of iterative min-sum decoding of cycle codes for the BSC.

Let $\mathcal{P}$ denote the set of NCICWs, and let $\mathcal{C}$ denote the set of cycles and disjoint union of cycles.

Recall if $\mathbf{p} = (p_1, \cdots, p_L)$ is a NCICW of length L, then c_i is the number of times a codeword node v_i appears in $\mathbf{p}$ for $i \in [n]$. For a BSC, the union bound for an iterative min-sum decoder word error probability $P_{E_{BSC}}^{\mathrm{IT}}$ is

$$P_{E_{BSC}}^{\mathrm{IT}} \leq \sum_{\mathbf{p} \in \mathcal{P} \cup \mathcal{C}} \sum_{i=1}^{\langle \mathbf{p} \rangle} q(\mathbf{p}, i) p^i (1-p)^{\langle \mathbf{p} \rangle - i}, \tag{6.5}$$

where $\langle \mathbf{p} \rangle$ be the number of nonzero elements in $\mathbf{c} = (c_1, c_2, \cdots, c_n)$. For a ML decoder, the standard union bound for the word error probability $P_{E_{BSC}}^{\mathrm{ML}}$ is

$$P_{E_{BSC}}^{\mathrm{ML}} \leq \sum_{\mathbf{p} \in \mathcal{C}} \sum_{i=1}^{\langle \mathbf{p} \rangle} q(\mathbf{p}, i) p^i (1-p)^{\langle \mathbf{p} \rangle - i}. \tag{6.6}$$

Note, for a codeword $\mathbf{p}$, with $\langle \mathbf{p} \rangle = t$, the ordinary weight of the codeword, and

$$\sum_{i=1}^{\langle \mathbf{p} \rangle} q(\mathbf{p}, i) p^i (1-p)^{\langle \mathbf{p} \rangle - i} = Q_t = \begin{cases} \sum_{r=(t+1)/2}^{t} \binom{t}{r} p^r (1-p)^{t-r} & \text{if } d \text{ is odd,} \\ Q_{t-1} & \text{if } d \text{ is even.} \end{cases}$$

Since the minimum pseudoweight of a bad NCICW is at least 2 times the minimum distance of the cycle code, these two union bounds are asymptotically equal.

6.4 Conclusions

For a cycle code, because the minimum pseudoweight of a bad NCICW is as least 2 times the minimum distance of the code, the performance of iterative min-sum

[5] It is the same proof as the union bound on the performance of iterative decoding of tail-biting codes in Chapter 5.

decoding will be asymptotically the same as ML decoding.

Chapter 7 Analysis of RA Codes under Message-Passing

In [16], a class of rate $1/q$ serially concatenated turbo codes, where the outer code is a q-fold repetition code and the inner code is a rate 1 convolutional code with transfer function $1/(1+D)$, is constructed. As the block length approaches infinity, the ensemble (over all possible interleavers) maximum likelihood error probability of this class of RA codes approaches zero if E_b/N_0 exceeds some threshold over AWGN channels. For $q = 3$, the threshold is no less than 0.792 dB.

The paper by Richardson and Urbanke [57] provides a general approach for determining the capacity of belief propagation decoding for low density parity check codes. By representing RA codes with appropriate Tanner graphs, we extend the analysis of [57] to RA codes, and we obtain the "threshold" values for belief propagation decoding of RA codes. If E_b/N_0 exceeds the threshold, the belief propagation decoding error probability approaches zero; whereas for E_b/N_0 below the threshold, the error probability stays bounded away from zero. For $q = 3$, the threshold is 0.776 dB on AWGN channels.

For $q = 3$ RA codes, with an information block length of $16,384$, simulated performance using belief propagation decoding is very close (< 0.8 dB) to the threshold value [16, Figure 5].

7.1 Introduction

Many innovative variations of turbo codes (parallel concatenated codes) have been discovered [9, 11, 12, 15, 26, 56] since the discovery of turbo codes in 1993.

The turbo decoding algorithm is now known to be an instance of Pearl's belief propagation algorithm [53] in [42, 33], and so is the decoding algorithm of another class

of capacity achieving codes, the low-density parity-check (LDPC)codes [25, 48, 49, 50].

It is well known that for cycle free graphs, belief propagation decoding results in a ML or a MAP decision [42, 33]. However, the best graphical representations of turbo and LDPC codes contain cycles. It is well known that belief propagation decoding on graphs with cycles is suboptimal. Nevertheless, there is a large body of empirical evidence illustrating the excellent performance of belief propagation decoding on turbo and LDPC codes. While a satisfactory theoretical explanation of why the belief propagation decoding algorithm performs as well as it does is still lacking, determining the channel *threshold value* δ^* [57] of belief propagation decoding sheds some light. A threshold value δ^* has the following properties: for a channel with parameter $\delta < \delta^*$, as the length of the code approaches infinity, almost all codes have error probability approaching zero (in the limit of the number of iterations tending to ∞). Conversely, if $\delta > \delta^*$, the error probability is bounded away from zero.

In [25], Gallager suggests a natural decoding algorithm and proves a good lower bound on the fraction of errors that can be corrected if the underlying graphs of regular LDPC codes[1] have no short cycles. While in practice LDPC codes are chosen in random, this analysis does not apply to such codes directly because a randomly chosen code may have short cycles.

Luby et al. develop an analysis that applies to randomly chosen LDPC codes[2] in [38]. Their analysis can be summarized in the following three steps. Step one: assume the underlying graphs have no short cycles and apply Gallager's analysis to estimate the expected fraction of errors at each iteration. Step two: construct a martingale and apply Asuma's inequality [47] to show that at most a fraction γ/n of all edges participate in cycles of small length for a randomly chosen code with length n. Consequently, for a randomly chosen graph and a randomly chosen input, with probability close to 1, the fraction of incorrect messages passed at the l-th decoding step is within $\varepsilon > 0$ of the estimated value. Step three: another algorithm, such as the one proposed by Spielman and Sipser [62], is used to correct the remaining errors

[1] In its original form, a LDPC codes has j ones and k ones respectively in each row and each column of the parity check matrix, and it is called a (j, k) regular LDPC code.

once the fraction of incorrect messages has been reduced to ε.

Richardson and Urbanke [57] extend the first step in Luby et al.'s analysis to a broad set of message passing decoding algorithms, which include the belief propagation algorithm. They provide an algorithm to determine the threshold value of belief propagation decoding on LDPC codes.

We represent a special family of serially concatenated turbo-like codes, namely, the *repeat and accumulate* (RA) codes, by appropriate Tanner graphs. We extend the analysis of [57] to RA codes and obtain the threshold values under belief propagation decoding.

The rest of the chapter is organized as follows. Section 2 briefly introduces RA codes. Tanner graph representations of RA codes are given in section 3. In section 4, we will apply Gallager's hard decision decoding algorithm to RA codes over a BSC, and we will show that this decoding algorithm is worse than no coding. In section 5, we will extend the analysis of [57] to RA codes, and we will present an algorithm to determine the threshold value of RA codes under belief propagation decoding.

7.2 RA Codes

In [16], a class of serially concatenated turbo codes [12], called *repeat and accumulate* (RA) codes is constructed. The encoder for a (qN, N) RA code is shown in Figure 7.1. An information block $U = (u_1, \cdots, u_N)$ of length N is repeated q times and output as a length qN sequence $X = (x_1, \cdots, x_{qN})$. The X block is permuted by a $qN \times qN$ interleaver, and the output of the interleaver $X' = (x'_1, \cdots, x'_{qN})$ is them encoded by a rate 1 *accumulator* with transform function $1/(1 + D)$.

It is shown in [16] that as the block length approaches infinity, the ensemble (over all possible interleavers) RA code maximum likelihood error probability approaches zero if E_b/N_o exceeds some threshold γ_0. The proof technique is to derive an explicit expression for the ensemble input-output weight enumerator, using the *uniform in-*

[2]This analysis is extended to irregular LDPC codes as well, which are proposed by Luby et al. and MacKay et al. Empirically, irregular LDPC codes have better performance than the regular LDPC codes [58, 51, 50, 37, 38].

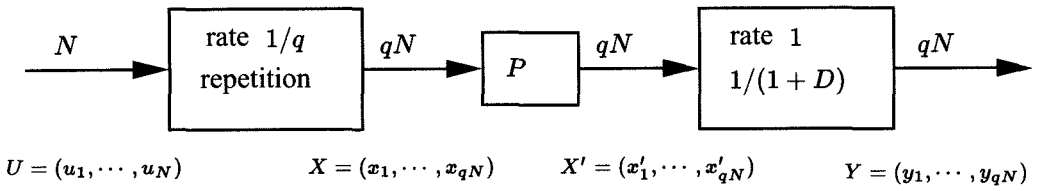


Figure 7.1: Encoder for a (qN, N) RA code.

terleaver technique [10]. This expression for the input-output weight enumerator, in combination with either the classical union bound, the union bound of Viterbi and Viterbi [68], or the recent improved geometry bound of "simplified Gallagar's bound" [18], [16] is used to show that the maximum likelihood word or bit error probability approaches zero as $N \to \infty$.

[16] lists the threshold γ_0 for RA codes with q in the range $3 \leq q \leq 8$ using both the classical union bound and the Viterbi-Viterbi union bound. In particular, for $q = 3$, the classical union bound gives $\gamma_0 = 2.200$ dB, whereas γ_0 improves to 1.112 dB using the Viterbi-Viterbi union bound. [16] points out that these values of γ_0 are by no means the best possible, and it is conjectured that they can be reduced further, for example by using the bound of [17]. In fact, γ_0 improves to 0.792 dB using the recent improved geometry bound of "simplified Gallagar's bound" [18].

7.3 Tanner Graph Representations of RA Codes

A Tanner graph [64, 69] $G = (V, E)$ is a bipartite graph consisting of a set of message nodes V_m and check nodes V_c, where $V = V_m \cup V_c$ with $V_m \cap V_c = \emptyset$ and $E \subseteq V_m \times V_c$.

The inner code of a RA code, the accumulator, can be viewed as a truncated rate-1 recursive convolutional encoder with transfer function $1/(1 + D)$. But we prefer to think of it as a block code whose input block $(x_1, \cdots, x_n)$ and output block $(y_1, \cdots, y_n)$ are related by the formula

$$y_i = \begin{cases} x'_1 & \text{if } i = 1, \\ x'_i \oplus y_{i-1} & \text{otherwise.} \end{cases} \tag{7.1}$$

Figure 7.2 shows a Tanner graph for a $q = 3$ RA code (with identity interleaver). The outer repetition code is represented by the connection from information nodes to check nodes, whereas the inner code is represented by the connection from check nodes to code nodes given by eqn. (7.1). In these Tanner graphs, the message nodes consists of information node $V_{m'}$ and code nodes V_m, i.e., $V_s = V_{m'} \cup V_m$.

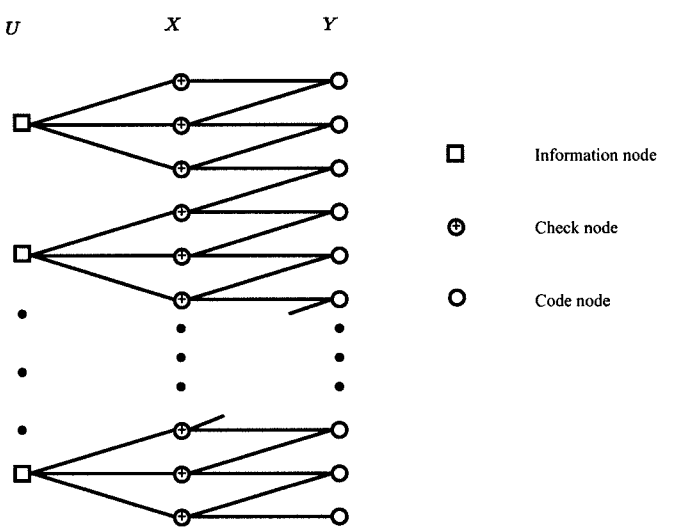


Figure 7.2: Tanner graph of a $q = 3$ RA code (with identity interleaver). The outer repetition code is represented by the connection from information nodes to check nodes, whereas the inner code is represented by the connection from check nodes to code nodes given by eqn. (7.1).

In the rest of this chapter, we will denote an information node by $m'_i, i \in [n/q]$, a code node by m_j, and a check node by c_j, $j \in [n]$.

7.4 Gallager's Algorithm Applied to RA Codes

In this section we will apply Gallager's hard decision decoding algorithm to RA codes over a BSC, and we will analyze the error probability of RA codes using this decoding algorithm.

7.4.1 Gallager's Decoding Algorithm

Gallager's hard decision decoding algorithm for decoding a (j, k) regular LDPC code is as follows. First the decoder computes all the parity checks and then changes any message node that is contained in more than some fixed number of unsatisfied check nodes. Using the new value on the message nodes, the parity checks are recomputed, and the decoding process is repeated until all the parity checks are satisfied.

Gallager's decoding is reasonable because most of the parity check nodes will contain either one transmission error or no transmission errors. Thus, there is a strong indication that a message node is in error when most of the parity checks by which the message node is checked are unsatisfied.

Gallager show that an arbitrarily small error probability for a rate $1/3$, $(4, 6)$ regular LDPC code is achievable if the crossover probability of the BSC is < 0.075.

7.4.2 Analysis of Gallager's Decoding Algorithm Applied to RA Codes

Consider the following modification of Gallager's algorithm for decoding a rate $1/q$ RA code. The decoder first computes all the parity checks, and it changes any code node that is contained in one of unsatisfied parity check nodes and any information node that is contained in $(q-1)$ of the unsatisfied parity check nodes. Using the new value on the information nodes and code nodes, the parity checks are recomputed, and the decoding process is repeated.

We now determine the probability that a code bit is in error when applying this decoding procedure assuming that the underlying graph (Figure 7.2) is cycle free.

Let the probability of a code node received in error be $0 < P_0 < 1/2$. Since information nodes are not transmitted, we will assume that the probability of an information node received in error is $P_0' = 1/2$. Let the error probability of a code (an information) node in the ith round of decoding be denoted by P_i (P_i').

A check node which constrains a code node received in errors will be unsatisfied if and only if an even number (including zero) of errors occur among the other code node and the information node which is incident on the check node. The probability of such an event is

$$\frac{1+(1-2P_0)(1-2P_0')}{2}.$$

By the same reasoning, the probability that a code node is received correctly but is changed because of an unsatisfied check node is

$$(1-P_0)\left[\frac{1-(1-2P_0)(1-2P_0')}{2}\right].$$

Thus, the error probability, P_1, of a code node in the first round of decoding is

$$\begin{aligned} P_1 = P_0 \; &- \; P_0\left[\frac{1+(1-2P_0)(1-2P_0')}{2}\right] \\ &+ \; (1-P_0)\left[\frac{1-(1-2P_0)(1-2P_0')}{2}\right]. \end{aligned}$$

By induction, after the ith round decoding, it is easy to show that P_{i+1} is

$$\begin{aligned} P_{i+1} \;&=\; P_0 - P_0\left[\frac{1+(1-2P_i)(1-2P_i')}{2}\right] \\ &\quad + \; (1-P_0)\left[\frac{1-(1-2P_i)(1-2P_i')}{2}\right], \\ &=\; \frac{1-(1-2P_i)(1-2P_i')}{2}. \end{aligned} \tag{7.2}$$

By the similar reasoning, P'_{i+1} is

$$P'_{i+1} = P'_0 - P'_0 \left[\frac{1+(1-2P_i)^2}{2}\right]^{q-1} + (1-P'_0)\left[\frac{1-(1-2P_i)^2}{2}\right]^{q-1}. \quad (7.3)$$

By substituting eqn. (7.3) into eqn. (7.2), P_{i+1} becomes

$$P_{i+1} = \frac{1}{2}\left\{1-(1-2P_i)\left\{\left[\frac{1+(1-2P_i)^2}{2}\right]^{q-1} - \left[\frac{1-(1-2P_i)^2}{2}\right]^{q-1}\right\}\right\}.$$

Thus

$$P_{i+1} \to \frac{1}{2} \quad \text{as} \quad i \to \infty.$$

The code node error probability approaches 1/2 if we use Gallager's decoding algorithm, regardless of the channel crossover probability. This intuitively makes sense because we assume information nodes with probability 1/2 being received in error. Thus a lot of check nodes will contain more than one transmission error, and there is no strong indication that a message node is in error when most of the check nodes checking on that message node are unsatisfied. Hence, the decoding algorithm breaks down.

Gallager's decoding algorithm is a bad idea for decoding RA codes on a BSC because it is worse than having no decoding at all.

7.5 Belief Propagation Decoding of RA Codes

In [57], Richardson and Urbanke present a general method for determining the capacity of message passing decoding applied to LDPC codes over any binary input memoryless channel in the following two steps.

1. Assuming that the code is cycle free, determine the average over all decoder

inputs behavior of the decoding algorithm. Assuming that the channel is characterized by one real parameter, determine the threshold value below which the expected fraction of decoding error converges to zero as the number of iterations grows.

2. A general concentration theorem showed that for almost all codes and almost all inputs the decoder behaves within ε of the expected behavior. From this concentration theorem it follows that (when the code length n is sufficiently large) almost all codes can transmit reliably up to the threshold value, but that the error probability stays bounded away from zero when code transmit above the threshold value.

We will extend the analysis in step 1 to RA codes.

7.5.1 Message-Passing Algorithms

Denote $\mathcal{C}_n^q$ to be the ensemble of $n!$ RA codes of length n and repetition q (see Figure 7.1). We asumme that a permutation is chosen at random with uniform probability. We can also visualize the ensemble $\mathcal{C}_n^q$ to be generated in the following manners. Label the n edges incident to the information nodes and the n check nodes in Figure 7.2 separately with the set $[n] := \{1, 2, \cdots, n\}$. Pick a permutation π_n at random. An edge i is then connected to check node $\pi_n(i)$, $i \in [n]$. Note that the connection between a check node and a code node does not change in the Tanner graph Figure 7.2 regardless of the permutation chosen.

Code nodes are transmitted to a channel with output alphabet $\mathcal{O}$. Information nodes are not transmitted, so we assume them to be transmitted to a separate "very" noise channel with output alphabet $\mathcal{O}'$. We will assume that the channel output alphabets are equal to the decoder input alphabets for the two channels.

The class of *message-passing* algorithms which we consider is the following. Every code node m_i, $i \in [n]$ has an associated received value R_i, which is a random variable taking values in the channel output alphabet $\mathcal{O}$; every information node m_i', $i \in [n/q]$ has an associated received value R_i' which takes values in $\mathcal{O}'$. An iteration of

message-passing consists of the following steps. First, each code node m_i sends to each neighboring check node c_j a *message* taking values in some *message alphabet* $\mathcal{M}$, and every check node c_j sends to each neighboring information node m'_i a message taking values in the message alphabet $\mathcal{M}'$. Then, each information node m'_i sends to each neighboring check node c_j a message in $\mathcal{M}'$, and each check node c_j sends to each neighboring code node m_i a message in $\mathcal{M}$.

The message from code node m_i (or information node m'_i) to check node c_j along the edge $e := (m_i, c_j)$ (or $e := (m'_i, c_j)$) depends on the received value R_i (or R'_i) and all the incoming messages from the edge connected to m_i (or m'_i) except the one along the edge e. The message from a check node c_j to a code node m_i (or an information node m'_i) along the edge $e := (c_j, m_i)$ (or $e := (c_j, m'_i)$) depends on all the incoming messages from the edge connected to c_j except the one along the edge e. The detailed message updating rule is described in Chapter 1 where we view it as an instance of the Generalized Distributive Law (GDL).

The message sent from a node along an edge e is required to exclude the message received along the edge e. This requirement of excluding the incoming message along the edge e is an an important property of good message-passing decoding. Most importantly, this requirement and the assumption that the code is cycle free ensure that messages are independent and make the analysis in subsection 7.5.2 possible. Belief propagation is one such message-passing algorithm.

To picture the decoding process, consider an edge $e := (m', c)$ between an information node m' and a check node c, and an associated computation tree describing a neighborhood of m' as shown in Figure 7.3. This tree is rooted at m'. The tree branches out from all the check nodes of m' excluding c. We will assume that the neighborhood of m' is a tree for some fixed number of iterations.

7.5.2 Notations

Let $d_m = 2$ denote the degree of a code node,

let $d_c = 3$ denote the degree of a check node, and

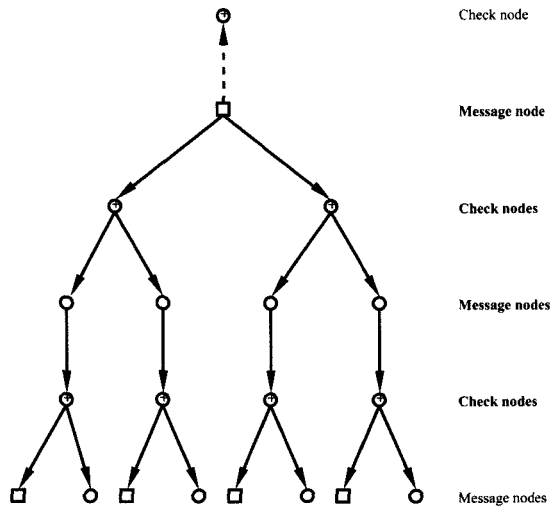


Figure 7.3: Computation tree of a $q = 3$ RA code after one iteration.

let $d_{m'} = q$ denote the degree of an information node.

For the l-th iteration, let $M_{code \to check}[l] : \mathcal{O} \times \mathcal{M}^{d_m - 1} \to \mathcal{M}$, $l \geq 1$, denote a code node to a check node message map,

let $M_{check \to info}[l] : \mathcal{M}^{d_c - 1} \to \mathcal{M}'$, $l \geq 0$, denote a check node to an information node message map,

let $M_{info \to check}[l] : \mathcal{O}' \times \mathcal{M}'^{d_{m'} - 1} \to \mathcal{M}'$, $l \geq 1$, denote an information node to a check node message map, and

let $M_{check \to code}[l] : \mathcal{M}' \times \mathcal{M} \to \mathcal{M}$, $l \geq 0$, denote a check node to a code node message map.

Furthermore, let $M_{code \to check}[0] = M^0_{code \to check} : \mathcal{O} \to \mathcal{M}$ denote the initial code node to a check node map, and

let $M_{info \to check}[0] = M^0_{info \to check} : \mathcal{O}' \to \mathcal{M}'$ denote the initial code node to a check node map.

Let Π_A denote the space of probability densities over any given set A.

We are interested in the evolution of message distributions during the execution

of the decoding algorithm. To this end, we define the following maps

$$\begin{aligned}
\tilde{M}_{code\to check}[l] &: \Pi_{\mathcal{O}} \times \Pi_{\mathcal{M}}^{d_m-1} \to \Pi_{\mathcal{M}}, \\
\tilde{M}_{check\to info}[l] &: \Pi_{\mathcal{M}}^{d_c-1} \to \Pi_{\mathcal{M}'}, \\
\tilde{M}_{info\to check}[l] &: \Pi_{\mathcal{O}'} \times \Pi_{\mathcal{M}'}^{d_{m'}-1} \to \Pi_{\mathcal{M}'}, \qquad \text{and} \\
\tilde{M}_{check\to code}[l] &: \Pi_{\mathcal{M}'} \times \Pi_{\mathcal{M}} \to \Pi_{\mathcal{M}}.
\end{aligned}$$

Note that $M^{-1}_{code\to check}[l][q]$ is the set of all inputs $(q_0, q_1, \cdots, q_{d_m-1})$ to a code node at step l that get mapped by the decoder to output q. Further, by the cycle free assumption, the probability density of the input being $(q_0, q_1, \cdots, q_{d_m-1})$ is $\prod_{i=0}^{d_m-1} P_i(q_i)$. Thus

$$\tilde{M}_{code\to check}[l](P_0, P_1, \cdots, P_{d_m-1})(q) = \sum \left\{ \prod_{i=0}^{d_m-1} P_i(q_i) : (q_0, q_1, \cdots, q_{d_m-1}) \in M^{-1}_{code\to check}[l](q) \right\},$$

where P_0 is a probability density over $\mathcal{O}$ and $P_1, \cdots, P_{d_m-1}$ are probability densities over $\mathcal{M}$. Similarly,

$$\tilde{M}_{check\to info}[l](P_1, \cdots, P_{d_c-1})(q) = \sum \left\{ \prod_{i=1}^{d_c-1} P_i(q_i) : (q_1, \cdots, q_{d_c-1}) \in M^{-1}_{check\to info}[l](q) \right\},$$

$$\tilde{M}_{info\to check}[l] : (P'_0, P'_1, \cdots, P'_{d_{m'}-1})(q) = \sum \left\{ \prod_{i=0}^{d_{m'}-1} P'_i(q_i) : (q'_0, q'_1, \cdots, q'_{d_{m'}-1}) \in M^{-1}_{info\to check}[l](q) \right\}, \qquad \text{and}$$

$$\tilde{M}_{check\to code}[l] : (P'_1, P_1)(q) = \sum \left\{ \prod P'_1 P_1 : (q'_1, q_1) \in M^{-1}_{check\to code}[l](q) \right\},$$

where P'_0 is a probability density over $\mathcal{O}'$ and $P'_1, \cdots, P'_{d_{m'}-1}$ are probability densities over $\mathcal{M}'$.

We will describe an efficient way of evaluating the density evolution in the following subsection.

7.5.3 Evaluation Density Evolution

Let the two pairs of non-negative reals (p_1, p_0) and (p'_1, p'_0) satisfying $p_1 + p_0 = 1$ and $p'_1 + p'_0 = 1$ be a probability density on a code node and an information node, respectively. We will assume that $\log \frac{p_1}{p_0}$ (log likelihood ratio) messages are being used. After receiving its messages, a node transmits to each neighboring node the conditional distribution of the message node conditioned on all information not coming from that particular neighboring node. We also assume that the code is cycle free in this stage of the analysis.

If $l_1, l_2, \cdots, l_k$ are likelihood ratios of conditional distributions of a given bit value conditioned on independent random variables, then the likelihood ratio of the node conditioned on all of the random variables is $\prod_{i=1}^{k} l_i$. Therefore, we have

$$M_{code\to check}(m_0, m_1, \cdots, m_{d_m-1}) := \sum_{i=0}^{d_m-1} m_i,$$

$$M_{info\to check}(m'_0, m'_1, \cdots, m'_{d_{m'}-1}) := \sum_{i=0}^{d_{m'}-1} m'_i.$$

Given the distributions $P_0, \cdots, P_{d_m-1}$ and $P'_0, \cdots, P'_{d_m-1}$ on the real quantities $m_0, \cdots, m_{d_m-1}$, and $m'_0, \cdots, m'_{d_{m'}-1}$ respectively, it follows that the distributions of $M_{code\to check}(m_0, m_1, \cdots, m_{d_m-1})$ and $M_{info\to check}(m'_0, m'_1, \cdots, m'_{d_{m'}-1})$ are the convolutions of those distributions, i.e.,

$$\tilde{M}_{code\to check}(P_0, P_1, \cdots, P_{d_m-1}) = P_0 \otimes P_1 \otimes \cdots \otimes P_{d_m-1},$$

$$\tilde{M}_{info\to check}(P'_0, P'_1, \cdots, P'_{d_m-1}) = P'_0 \otimes P'_1 \otimes \cdots \otimes P'_{d_m-1},$$

where $\otimes$ denotes convolution. Let $\mathcal{F}$ denote the Fourier transform. We then have

$$\tilde{M}_{code\to check}(P_0, P_1, \cdots, P_{d_m-1}) = \mathcal{F}^{-1}\left(\mathcal{F}(P_0)\mathcal{F}(P_1)\cdots\mathcal{F}(P_{d_m-1})\right),$$
$$\tilde{M}_{info\to check}(P'_0, P'_1, \cdots, P'_{d_m-1}) = \mathcal{F}^{-1}\left(\mathcal{F}(P'_0)\mathcal{F}(P'_1)\cdots\mathcal{F}(P'_{d_m-1})\right).$$

Let us consider the distribution of a message send from a check node c_j to an information node m'_i. There are $d_c - 1$ independent incoming messages $\log\frac{p_1^k}{p_0^k}$ along the edge (excluding e) incident on c_j, then the outgoing message along edge e is $\log\frac{p'_1}{p'_0}$, where p'_0 (or p'_1) is the probability that the product of these incoming messages has value 0 (or 1). The value p'_0 (or p'_1) is the probability that the mod 2 sum of the $(d_c - 1)$ independent random variables is equal to 0 (or 1).

Claim 1. *The probability vector (p_0, p_1) is given by the cyclic convolution of the $(d_c - 1)$ probability vectors (p_0^k, p_1^k), for $k \in [d_c - 1]$.*

Proof: We will only prove the $d_c = 3$ case here. The extension to $d_c > 3$ is straightforward but lengthy. Let $\breve{\otimes}$ denote cyclic convolution, then

$$\begin{aligned}(p_0^1, p_1^1) \;\breve{\otimes}\; (p_0^2, p_1^2) &= (p_0^1p_0^2 + p_1^1p_1^2, p_0^1p_1^2 + p_1^1p_0^2)\\ &= (p_0, p_1) \qquad\qquad \square\end{aligned}$$

Claim 2. *Let P denote the probability distribution of the parity check which checks bits with distributions p^k, for $k \in [d_c - 1]$. Then $P_0 - P_1 = \prod_{k=1}^{d_c-1}(p_0^k - p_1^k)$.*

Proof:

$$\begin{pmatrix}\mathcal{F}(P)(0)\\ \mathcal{F}(P)(1)\end{pmatrix} = \begin{pmatrix}P_0 & P_1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$
$$= \begin{pmatrix}P_0 + P_1\\ P_0 - P_1\end{pmatrix}.$$

Then,

$$\begin{aligned}(\mathcal{F}(P)(0), \mathcal{F}(P)(1)) &= \prod_{k=1}^{d_c-1} \left(\mathcal{F}(p^k)(0), \mathcal{F}(p^k)(1)\right) \\ P_0 - P_1 &= \prod_{k=1}^{d_c-1} (p_0^k - p_1^k).\end{aligned}$$

Note $\mathcal{F}(P)(0) = P_0 + P_1 = 1$ □

Let m denote the log-likelihood ratio $\log \frac{p_1}{p_0}$, and let q denote $p_0 - p_1$. Then

$$\log \frac{p_1}{p_0} = \frac{1-q}{1+q} = \frac{1-(p_0-p_1)}{1+(p_0-p_1)}.$$

Thus,

$$M_{check \to info}(m_1, \cdots, m_{d_c-1}) := \log \left(\frac{1 - \prod_{i=1}^{d_c-1} \frac{1-e^{m_i}}{1+e^{m_i}}}{1 + \prod_{i=1}^{d_c-1} \frac{1-e^{m_i}}{1+e^{m_i}}} \right).$$

Similarly, let m' denote the log-likelihood ratio $\log \frac{p_1'}{p_0'}$, we have

$$M_{check \to code}(m_1', m_1) := \log \left(\frac{1 - \left(\frac{1-e^{m_1}{}'}{1+e^{m_1}{}'}\right)\left(\frac{1-e^{m_1}}{1+e^{m_1}}\right)}{1 + \left(\frac{1-e^{m_1}{}'}{1+e^{m_1}{}'}\right)\left(\frac{1-e^{m_1}}{1+e^{m_1}}\right)} \right).$$

To determine the distribution of $\prod_{i=1}^{k} q_i$, where $q_i = \left(\frac{1-e^{m_i}}{1+e^{m_i}}\right)$ and supported on $(-1, 1)$. We will follow [57] and represent q as the ordered pair $(-\log|q|, \operatorname{lgsgn} q) \in [0, \infty) \times \mathbb{Z}/2$, where

$$\operatorname{lgsgn} q = \begin{cases} 1 & \text{if } q < 0, \\ 0 & \text{if } q > 0. \end{cases}.$$

Let Q denote the distribution of $(-\log|q|, \operatorname{lgsgn} q)$ and R denote the distribution of $(-\log|r|, \operatorname{lgsgn} r) = \sum_{i-1}^{k} (-\log|q_i|, \operatorname{lgsgn} q_i)$. Then R is the convolution of Q's, and the densities $\tilde{M}_{check \to info}(m_1, \cdots, m_{d_c-1})$ and $\tilde{M}_{check \to code}(m_1', m_1)$ can be obtained from R by a change of variables.

Define $Q^0(y) := Q(y, 0)$ and $Q^1(y) := Q(y, 1)$, $y \in [0, \infty)$. Then the Fourier

transform of Q is given by

$$\begin{aligned}\mathcal{F}(Q)(s,0) &= \mathcal{F}\left(Q^0(s)\right) + \mathcal{F}\left(Q^1(s)\right), \\ \mathcal{F}(Q)(s,1) &= \mathcal{F}\left(Q^0(s)\right) - \mathcal{F}\left(Q^1(s)\right).\end{aligned}$$

Then the Fourier transform of R is given by

$$\begin{aligned}\mathcal{F}(R)(s,0) &= \prod_{i=0}^{d_c-1}\left(\mathcal{F}\left(Q_i^0(s)\right) + \mathcal{F}\left(Q_i^1(s)\right)\right), \\ \mathcal{F}(R)(s,1) &= \prod_{i=0}^{d_c-1}\left(\mathcal{F}\left(Q_i^0(s)\right) - \mathcal{F}\left(Q_i^1(s)\right)\right).\end{aligned}$$

The rest of this subsection is devoted to obtaining Q and S, the distribution of $\log\frac{1-\prod_{i=1}^{d_c-1}q_i}{1+\prod_{i=1}^{d_c-1}q_i}$. Let P denote the density associated with m. Write $P = P^0 + P^1$, where P^0 is supported on $(-\infty, 0]$ and P^1 is supported on $[0, \infty)$.

Claim 3. *Let $y = -\log|\frac{1-e^m}{1+e^m}|$, where m and y are distributed according to P and Q, respectively. Then*

$$Q^0(y) = \left(\frac{2e^y}{e^{2y}-1}\right) P^0\left(\log\frac{e^y-1}{e^y+1}\right) \tag{7.4}$$

$$Q^1(y) = \left(\frac{2e^y}{e^{2y}-1}\right) P^1\left(-\log\frac{e^y-1}{e^y+1}\right) \tag{7.5}$$

for $y \geq 0$.

Proof: For $m \leq 0$, $y = -\log\left(\frac{1-e^m}{1+e^m}\right)$. Then

$$\begin{aligned}m &= \log\frac{e^y-1}{e^y+1}, \text{ and} \\ \frac{dm}{dy} &= \frac{2e^y}{e^{2y}-1}.\end{aligned}$$

Thus we have

$$\begin{aligned} Q^0(y) &= \frac{P^0(m)}{dy/dm} \\ &= \left(\frac{2e^y}{e^{2y}-1}\right) P^0\left(\log\frac{e^y-1}{e^y+1}\right). \end{aligned}$$

Similarly, for $m > 0$, $y = -\log\left(-\frac{1-e^m}{1+e^m}\right)$. Then

$$\begin{aligned} m &= -\log\frac{e^y-1}{e^y+1}, \text{ and} \\ \frac{dm}{dy} &= \frac{-2e^y}{e^{2y}-1}. \end{aligned}$$

Thus we have

$$\begin{aligned} Q^1(y) &= \frac{P^1(m)}{-dy/dm} \\ &= \left(\frac{2e^y}{e^{2y}-1}\right) P^1\left(-\log\frac{e^y-1}{e^y+1}\right). \end{aligned}$$ □

Claim 4. *Let*

$$y = \begin{cases} -\log\frac{1-e^{-r}}{1+e^{-r}} & \text{if } r \geq 0, \\ \log\frac{1+e^{-r}}{1-e^{-r}} & \text{if } r < 0. \end{cases},$$

where r and y are distributed according to R and S. Then

$$\begin{aligned} S^0(-y) &= \left(\frac{2e^y}{e^{2y}-1}\right) R^0\left(\log\frac{e^y+1}{e^y-1}\right) \\ S^1(y) &= \left(\frac{2e^y}{e^{2y}-1}\right) R^1\left(\log\frac{e^y+1}{e^y-1}\right) \end{aligned}$$

for $y \geq 0$.

Proof: For $r \geq 0$, $y = -\log \frac{1-e^{-r}}{1+e^{-r}}$. Then

$$\begin{aligned} \log r &= \log \frac{e^y+1}{e^y-1}, \text{ and} \\ \frac{dr}{dy} &= \frac{-2e^y}{e^{2y}-1}. \end{aligned}$$

Thus we have

$$\begin{aligned} S^0(-y) &= \frac{R^0(r)}{-dy/dr} \\ &= \left(\frac{2e^y}{e^{2y}-1}\right) R^0\left(\log \frac{e^y+1}{e^y-1}\right). \end{aligned}$$

Similarly, for $r < 0$, we have

$$S^1(y) = \left(\frac{2e^y}{e^{2y}-1}\right) R^1\left(\log \frac{e^y+1}{e^y-1}\right). \qquad \square$$

We summarize on how to update densities for belief propagation for RA codes in the next subsection.

7.5.4 An Algorithm for Updating Densities for RA Codes Under Belief Propagation

Let P_l (or P_l') denote the common density associated with the messages from code (or information) nodes to check nodes in the lth iteration, and let P_0 (or P_0') denote the density of the received values at code (or information) nodes.

Write $P_l = P_l^0 + P_l^1$, $P_l' = P_l^{0'} + P_l^{1'}$, where P_l^0, $P_l^{0'}$ are supported on $(-\infty, 0]$ and P_l^1, $P_l^{1'}$ are supported on $[0, \infty)$. First we find the density Q_l of $(-\log|q|, \operatorname{lgsgn} q)$ where $q = \frac{1-e^m}{1+e^m}$, and the density Q_l' of $(-\log|q'|, \operatorname{lgsgn} q')$ where $q' = \frac{1-e^{m'}}{1+e^{m'}}$. The distribution of m (or m') is according to P_l (or P_l').

Representing Q_l by (Q_l^0, Q_l^1), where both Q_l^0 and Q_l^1 are supported on $[0, \infty)$, we

have

$$Q_l^0(y) = \left(\frac{2e^y}{e^{2y}-1}\right) P_l^0\left(\log\frac{e^y-1}{e^y+1}\right) = (\operatorname{csch} y)\, P_l^0\left(\log\tanh\frac{y}{2}\right), \tag{7.6}$$

$$Q_l^1(y) = \left(\frac{2e^y}{e^{2y}-1}\right) P_l^1\left(-\log\frac{e^y-1}{e^y+1}\right) = (\operatorname{csch} y)\, P_l^1\left(\log\tanh\frac{y}{2}\right). \tag{7.7}$$

for $y \geq 0$.

Similarly, representing Q_l' by $(Q_l^{0'}, Q_l^{1'})$, where both $Q_l^{0'}$ and $Q_l^{1'}$ are supported on $[0, \infty)$, we have

$$Q_l^{h0}(y) = (\operatorname{csch} y)\, P_l^{h0}\left(\log\tanh\frac{y}{2}\right), \tag{7.8}$$

$$Q_l^{h1}(y) = (\operatorname{csch} y)\, P_l^{h1}\left(\log\tanh\frac{y}{2}\right). \tag{7.9}$$

Next, we want to derive the probability densities R_l and R_l', which are the densities of random variables given by the sum of $d_c - 1$ independent random variables distributed respectively according to Q_l or Q_l'. Let (R_l^0, R_l^1) represent R_l, and $(R_l^{0'}, R_l^{1'})$ represent R_l', we then have

$$\mathcal{F}(R_l^0) - \mathcal{F}(R_l^1) = \left(\mathcal{F}(Q_l^0) - \mathcal{F}(Q_l^1)\right)\left(\mathcal{F}(Q_l^{0'}) - \mathcal{F}(Q_l^{1'})\right), \tag{7.10}$$

$$\mathcal{F}(R_l^0) + \mathcal{F}(R_l^1) = \left(\mathcal{F}(Q_l^0) + \mathcal{F}(Q_l^1)\right)\left(\mathcal{F}(Q_l^{0'}) + \mathcal{F}(Q_l^{1'})\right), \tag{7.11}$$

$$\mathcal{F}(R_l^{0'}) - \mathcal{F}(R_l^{1'}) = \left(\mathcal{F}(Q_l^0) - \mathcal{F}(Q_l^1)\right)^2, \tag{7.12}$$

$$\mathcal{F}(R_l^{0'}) + \mathcal{F}(R_l^{1'}) = \left(\mathcal{F}(Q_l^0) + \mathcal{F}(Q_l^1)\right)^2. \tag{7.13}$$

Now, let S_l (or S_l') denote the common density associated with the message from

check nodes to variable nodes (or information nodes). We have

$$S_l^0(-y) = \left(\frac{2e^y}{e^{2y}-1}\right) R_l^0\left(\log\frac{e^y+1}{e^y-1}\right), \tag{7.14}$$

$$S_l^1(y) = \left(\frac{2e^y}{e^{2y}-1}\right) R_l^1\left(\log\frac{e^y+1}{e^y-1}\right), \tag{7.15}$$

$$S_l^{0'}(-y) = \left(\frac{2e^y}{e^{2y}-1}\right) R_l^{0'}\left(\log\frac{e^y+1}{e^y-1}\right), \tag{7.16}$$

$$S_l^{1'}(-y) = \left(\frac{2e^y}{e^{2y}-1}\right) R_l^{1'}\left(\log\frac{e^y+1}{e^y-1}\right). \tag{7.17}$$

Thus,

$$\mathcal{F}(P_{l+1}) = \mathcal{F}(P_0)\mathcal{F}(S_l), \tag{7.18}$$

$$\mathcal{F}(P'_{l+1}) = \mathcal{F}(P'_0)\left(\mathcal{F}(S'_l)\right)^2. \tag{7.19}$$

7.5.5 Thresholds for Belief Propagation

We will apply the above algorithm to the binary-input AWGN channel with belief propagation on RA codes.

Let x be the AWGN channel output. We have

$$P_1 = \frac{1}{\sqrt{2\pi\sigma}}\exp^{\frac{-(x+1)^2}{2\sigma^2}}, \text{ and}$$

$$P_0 = \frac{1}{\sqrt{2\pi\sigma}}\exp^{\frac{-(x-1)^2}{2\sigma^2}}.$$

Let $y = \log\frac{P_1}{P_0} = -\frac{2}{\sigma^2}x$, then

$$P_0(y) = \frac{\sigma}{2\sqrt{2\pi}}\exp^{\frac{-(y+\frac{2}{\sigma^2})^2}{8/\sigma^2}}.$$

Since information nodes do not transmit to the channel, we require $\log\frac{P'_1}{P'_0} = 0$. Equivalently,

$$P'_0(y) = \begin{cases} 1 & \text{if } y = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Applying the algorithm in previous section using the P_0 and P_0', we obtain the threshold values for $q = 3$ and $q = 4$ RA codes under belief propagation. The threshold values are shown in the table below, and for comparison we also include the threshold values for maximum likelihood decoding. The maximum likelihood threshold values are very close to the belief propagation threshold values. Figure 5 of [16] shows a simulated performance of belief propagation < 0.8 dB using an information block of length $16,384$.

q	3	4
Maximum Likelihood (E_b/N_0 (dB))	≤ 0.792	≤ -0.052
Belief Propagation (E_b/N_0 (dB))	0.776	-0.096
Binary Shannon Limit (E_b/N_0 (dB))	-0.495	-0.794

7.6 Conclusions

As the block length approaches infinity, the ensemble (over all possible interleavers) maximum likelihood error probability of the class of RA codes approaches zero if E_b/N_0 exceeds some threshold over AWGN channels. For $q = 3$, the threshold is no less than 0.792 dB.

By applying Gallager's hard decision decoding algorithm on RA codes over a BSC, error probability approaches $1/2$ regardless of the channel crossover probability. Thus, Gallager's decoding algorithm is a bad idea for decoding RA codes on a BSC because it is worse than having no decoding.

By representing RA codes with appropriate Tanner graphs, we extend the analysis of [57] to RA codes, and we obtain the threshold values for belief propagation decoding of RA codes. If E_b/N_0 exceeds the threshold, the belief propagation decoding error probability approaches zero; whereas for E_b/N_0 below the threshold, the error probability stays bounded away from zero. For $q = 3$, the threshold is 0.776 dB on AWGN channels.

Chapter 8 Conclusion

This thesis examines the iterative decoding of codes defined on graphs with cycles, which appears to be an efficient means of achieving the Shannon limit. Much of the analysis has been focussed on the iterative min-sum decoding of tail-biting codes and cycle codes. We have identified the pseudocodeword as the cause of the suboptimum performance of the iterative decoder, and we have obtained a union bound for the performance of the iterative min-sum decoder on both AWGN and BSC channel.

Using the union bound argument, for cycle codes we have shown that the performance of the iterative min-sum decoder is asymptotically as good as the ML decoder. For tail-biting codes, if the lowest weight pseudocodeword is at least as large as the minimum weight of the code, the iterative min-sum decoder is asymptotically as good as an ML decoder. For a tail-biting trellis with a minimum state complexity of two, its pseudoweight will not be less than the minimum weight of the code. However, this is not true for a tail-biting trellis whose state complexity is three or more, and it remains a challenging problem to produce a practical algorithm for computing the minimum pseudoweight.

Unfortunately, the analysis of tail-biting codes and cycle codes does not extend to turbo codes and low density parity check codes in general. For these codes, the pseudocodewords have a tree-like structure in the computation tree, and the number of leaves grows exponentially with the depth of the tree. Thus a significant portion of weight will lie in the leaves, and pseudocodewords, unlike the case for tail-biting codes and cycle codes, will not be the determining factors in decoding errors [69].

Our next approach is to determine the average behavior of message passing algorithms by studying the evolution of their "message" densities. Applying the theorem in [57] which shows that for all most all codes and almost all inputs, the decoder behave within ϵ of its average behavior, we have devised an algorithm for obtaining the threshold values for the class of "repeat and accumulate" serially concatenated

turbo-like codes. When the channel's signal to noise ratio is larger than the threshold value, the error probability of the message passing algorithms approaches zero, whereas if the signal-to-noise ratio is less than the threshold value, the error probability stays bounded away from zero. The extension of this analysis to more general turbo codes is a topic for future research.

In the course of studying message passing algorithms and graphical representation of codes, we came to discover that some message passing algorithms and graphical representation of codes are efficient means of devising exact (ML or MAP) decoding algorithms. In particular, by applying the Generalized Distributive Law, we have proposed a junction tree representation for linear block codes, and we have shown that minimum junction trees can be less complex than the minimal trellises. We have also provided an algorithm for finding a minimum junction tree.

Bibliography

[1] S. M. Aji, *Graphical Models and Iterative Decoding.* PhD. Thesis, Dept. Elec. Eng., California Institute of Technology, Pasadena, CA, 1998.

[2] S. M. Aji, G. B. Horn, R. J. McEliece, and M. Xu, "Iterative min-sum decoding of tail-biting codes," *Proc. IEEE Information Theory Workshop*, Killarney Ireland (June 1998), pp. 68-69.

[3] S. M. Aji and R. J. McEliece, "The generalized distributive law," accepted for publication by *IEEE Trans. Inform. Theory.*

[4] S. M. Aji, G. B. Horn, R. J. McEliece, and M. Xu, "Iterative min-sum decoding of tail-biting codes," *Proc. 36th Allerton Conference on Communication, Control, and Computing* (Urbana, Illinois, September 1998), pp. 15-24.

[5] J. B. Anderson and S. M. Hladik, "Tail-biting MAP decoders," *IEEE J. Select. Areas Comm.*, vol. 16, no. 2 (February 1998), pp. 297-302.

[6] J. B. Anderson, personal communication.

[7] L. R. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20 (March 1974), pp. 284-287.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," *Proc. 1993 IEEE International Conference on Communications*, Geneva, Switzerland (May 1993), pp. 1064-1070.

[9] A. S. Barbulescu and S. S. Pietrobon, "Interleaver Design for Three Dimensional Turbo-Codes," *Proc. 1995 IEEE International Symposium on Information Theory*, Whistler, British Columbia, Canada (September 1995), pp. 37.

[10] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes," *IEEE Trans. Inf. Theory*, vol. IT-42, no. 2 (March 1996), pp. 409-428.

[11] S. Benedetto and G. Montorsi, "Serial Concatenation of Block and Convolutional Codes," *Electronics Letters*, vol. 32, no. 10 (May 1996), pp. 887-888.

[12] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Trans. Information Theory*, vol. IT-44, no. 3, (May 1998), pp. 909-926.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, Mass.: MIT-McGraw-Hill, 1990.

[14] A. R. Calderbank, G. D. Forney, Jr., and A. Vardy, "Minimal tail-biting trellises: the Golay code and more," to appear in *IEEE Trans. Inform. Theory.*

[15] J.-F. Cheng and R. J. McEliece, "Unit Memory Hamming Turbo Codes," *Proc. 1995 IEEE International Symposium on Information Theory*, Whistler, British Columbia, Canada (September 1995), pp. 3.

[16] D. Divsalar, H. Jin, and R. J. McEliece, "Coding Theorems for "Turbo-Like" Codes," *Proc. 36th Allerton Conference on Communication, Control, and Computing* (Urbana, Illinois, September 1998), pp. 201-210.

[17] S. Dolinar, L. Ekroot, and F. Pollara, "Improved Error Probability Bounds for Block Codes for the Gaussian Channel," *Proc. 1994 IEEE International Symposium on Information Theory*, pp. 43.

[18] D. Divsalar, "Simplified Gallager's Bound," Preprint.

[19] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. IT-61 (March 1973), pp. 268-276.

[20] G. D. Forney, Jr. "Coset Codes III: Sequential decoding," *Inform. Control*, vol. 25 (1974), pp. 267-297.

[21] G. D. Forney, Jr., F. R. Kschischang, and B. Marcus, "Iterative decoding of tail-biting trellises," Proc. 1998 Information Theory Workshop (San Diego: Feb. 9–11, 1998), pp. 11-12.

[22] G. D. Forney, Jr., F. R. Kschischang, and A. Reznik, "The effective weight of pseudocodewords for codes defined on graphs with cycles on AWGN channels," Preprint.

[23] B. J. Frey, F. R. Kschischang, H.-A. Loeliger and N. Wiberg, "Factor graphs and the sum-product algorithm," submitted to *IEEE Trans. Inf. Theory*.

[24] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inf. Theory*, vol. IT-9 (1963), pp. 64-73.

[25] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, Mass.: MIT Press, 1963.

[26] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Information Theory*, vol. IT-42 (March 1996), pp. 429-445.

[27] G. B. Horn, *Iterative Decoding and Pseudo-Codewords*, PhD. Thesis, Dept. Elec. Eng., California Institute of Technology, Pasadena, CA, 1999.

[28] F. V. Jensen, *An Introduction to Bayesian Networks*. New York: Springer-Verlag, 1996.

[29] F. V. Jensen and F. Jensen, "Optimal junction trees," pp. 360-366 in *Proc. 10th conf. on Uncertainty in Artificial Intelligence*, R.L. de Mantaras and D. Poole, eds. San Francisco: Morgan Kaufmann, 1994.

[30] D. Jungnickel and S. A. Vanstone, "Graphical Codes Revisited," *IEEE Trans. Inf. Theory*, vol. 43 (January 1997), pp. 136-146.

[31] R. Köetter, and A. Vardy, personal communication.

[32] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes," *IEEE Trans. Inform. Theory*, vol. IT-39 (1993), pp. 284-287.

[33] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Sel. Area Comm.* vol. 16, no. 2 (February 1998), pp. 219-230.

[34] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[35] W. Lin, *The Trellis Complexity of Block and Convolutional Codes.* PhD. Thesis, Dept. Elec. Eng., California Institute of Technology, Pasadena, CA, 1998.

[36] D. Lind and B. Marcus, *Symbolic Dynamics and Coding.* Cambridge, England: Cambridge University Press, 1995.

[37] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation," *Proc. 1998 IEEE International Symposium on Information Theory*, Cambridge, MA (August 1998), pp. 117.

[38] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, "Analysis of Low Density codes and Improved Designs Using Irregular Graphs," *Proceedings of the 30th ACM STOC*, May 1998.

[39] F. J. MacWilliams and N. J. A. Slonne, *The Theory of Error-Correcting Codes*, 1977.

[40] R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, 1977.

[41] R. J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-42 (July 1996), pp. 1072-1092.

[42] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE J. Sel. Area Comm.* vol. 16, no. 2 (February 1998), pp. 140-152.

[43] R. J. McEliece and M. Xu, "Junction Tree Representations for Linear Block Codes," *Proc. 32nd Conference on Information Sciences and Systems* (Princeton, N.J., March 1998), pp. 435-438.

[44] S. M. Aji, R. J. McEliece and M. Xu, "Viterbi's Algorithm and Matrix Multiplications," *Proc. 33rd Conference on Information Sciences and Systems* (Baltimore, Maryland, March 1999).

[45] R. J. McEliece and M. Xu, "Junction Tree Representations for Linear Block Codes," *Proc. 1998 IEEE International Symposium on Information Theory*, Cambridge, MA (August 1998), pp. 253.

[46] D. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, vol. IT-34 (September 1988), pp. 1049-1053.

[47] R. Motwani and P. Raghawan, *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995.

[48] D. J. C. MacKay and R. Neal, "Good Codes Based on Very Sparse Matrices," pp. 100-111 in *Proc. 5th IMA Conference on Cryptography and Coding* Colin Boyd, ed. (Springer Lecture Notes in Computer Science, vol. 1025, 1995).

[49] D. J. C. MacKay and R. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letters*, vol. 32 (August 1996) pp. 1645-1646.

[50] D. J. C. MacKay ,"Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. IT-45, no. 2 (March 1999), pp. 399-431.

[51] D. J. C. MacKay, S. Wilson, and M. Davey, "Comparison of Constructions of Irregular Gallager Codes," *Proc. 36th Allerton Conference on Communication, Control, and Computing* (Urbana, Illinois, September 1998), pp. 220-229.

[52] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, 1993.

[53] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publishers, 1988.

[54] V. S. Pless, and W. C. Huffman *Handbook of Coding Theory*, Elsevier, 1998.

[55] J. G. Proakis, *Digital Communications*, 2nd ed., McGraw-Hill, 1989.

[56] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near Optimum Decoding of Product Codes," in *Proceedings of GLOBECOM '94*, San Francisco, California, Nov. 1994, vol. 1, pp. 339-343.

[57] T. Richardson, and R. Urbanke, "The Capacity of Low-Density Parity Check Codes under Message-Passing Decoding," submitted to *IEEE Trans. Inf. Theory.*

[58] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of Provably Good Low-Density Parity Check Codes," submitted to *IEEE Trans. Inf. Theory.*

[59] G. Solomon and H. C. A. Van Tilborg, "A connection between block and convolutional codes." *SIAM J. Appl. Math.*, pp. 358–367, October 1979.

[60] C. E. Shannon, "A Mathematical Theory of Communication," *Bell system Technical Journal*, 1948.

[61] G. R. Shafer and P. P. Shenoy, "Probability propagation," *Annals of Math. and Art. Intel.*, vol. 2 (1990), pp. 327-352.

[62] M. Siper and D. A. Spielman, "Expander Codes," *IEEE Trans. Inf. Theory*, vol. IT-42(November 1996), pp. 1710-1722.

[63] M. K. Simon, S. M. Hinedi, and W. C. Lindsey, *Digital Communication Techniques: Signal Design and Detection*, Prentice Hill, 1995.

[64] R. M. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Trans. Inf. Theory*, vol. IT-27 (September 1981), pp. 533-547.

[65] A. Vardy, and F. R. Kschischang, "Proof of a Conjecture of McEliece Regarding the Expansion Index of the Minimal Trellis," *IEEE Trans. Inf. Theory*, vol. IT-42 (November 1996), pp. 2027-2034.

[66] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13 (April 1967), pp. 260-269.

[67] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, 1979.

[68] A. J. Viterbi and A. M. Viterbi, "An Improved Union Bound for Binary-input Linear Codes on the AWGN Channel, with Applications to Turbo Decoding," *Proc. 1998 IEEE International Symposium on Information Theory*, Cambridge, MA (August 1998), pp. 29.

[69] N. Wiberg, *Codes and Decoding on General Graphs*, PhD. Thesis, Dept. Elec. Eng., Linköping Univ., Sweden, 1996.

[70] N. Wiberg, H.-A. Loeliger and R. Kötter,"Codes and Iterative decoding on General Graphs," *Europ. Trans. Telecomm.*, vol. 6 (September/October 1995), pp. 513-525.

[71] Y. Weiss, "Correctness of local probability propagation in graphical models with loops," Submitted to *Neural Computation.*