UNIVERSITY OF CALIFORNIA

Los Angeles

Research on
Low-Density Parity Check Codes

A dissertation submitted in partial satisfaction

of the requirement for the degree Doctor of Philosophy

in Electrical Engineering

by

Tao Tian

B.S., Tsinghua University, 1999

M.S., University of California, Los Angeles, 2000

Ph.D., University of California, Los Angeles, 2003

2003

The dissertation of Tao Tian is approved.

Richard D. Wesel

Gregory J. Pottie

Adnan Darwiche

John D. Villasenor, Committee Chair

University of California, Los Angeles
2003

Contents

List of Figures

List of Tables

Acknowledgments

I wish to express my appreciation to Chris Jones, for the enlightening discussion between us. I would also like to thank Ksenija Lakovic and Michael Smith for their reviewing this work. Finally, I would like to dedicate this to my dear mom and dad. It is their unselfish support through everyday of my life, that make this possible.

VITA

January 20, 1976	Born, Chongqing, China
1999	B.E., Electrical Engineering Tsinghua University Beijing, China
2000	M.S., Electrical Engineering University of California, Los Angeles
1999-2003	Graduate Student Researcher Electrical Engineering Department University of California, Los Angeles
2002	Visiting Scholar Electrical Engineering Department University of Delaware, Newark, DE

PUBLICATIONS

- C. Jones, A. Matache, **T. Tian**, J. Villasenor, and R. Wesel. The Universality of LDPC Codes on Wireless Channels. *Military Communications Conference,October 2003.*

- **T. Tian**, J. Garcia-Frias, and W. Zhong. Density Evolution Analysis of Correlated Sources Compressed with LDPC Codes. *International Symposium on Information Theory, June 2003.*

- **T. Tian**, C. Jones, John Villasenor, and Rick Wesel. Construction of Irregular LDPC Codes with Low Error Floors. *IEEE International Conference on Communications, May 2003.*

- **T. Tian**, J. Garcia-Frias, and W. Zhong. Compression of Correlated Sources Using LDPC Codes. *Data Compression Conference, February 2003.*

- C. Jones, **T. Tian**, A. Matache, R. Wesel, and J. Villasenor. Robustness of LDPC Codes on Periodic Fading Channels. *IEEE GlobeCom, November 2002.*

- A. Li, J. Fahlen, **T. Tian**, L. Boloni, S. Kim, J. Park, and J. Villasenor. Generic Uneven Level Protection Algorithm for Multimedia Data Transmission over Packet-Switched Networks. *IEEE International Conference on Computer Communications and Networks Proceedings*, pp. 340-346 , October 2001.

- **T. Tian**, A. Li, J. Wen, and J. Villasenor. Priority Dropping in Network Transmission of Scalable Video. *IEEE International Conference on Image Processing Proceedings*, Volume III, pp. 400-403, September 2000.

ABSTRACT OF THE DISSERTATION

Research on

Low-Density Parity Check Codes

by

Tao Tian

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2003

Professor John D. Villasenor, Chair

Low-density parity-check (LDPC) codes are a family of codes proven to have good asymptotic ensemble properties. There are many open theoretical and practical issues related to LDPC codes such as how to construct finite length LDPC codes with guaranteed properties, how to realize rate compatibility with these codes, and how to apply LDPC codes to source coding. This contribution first explains the relationship between cycles, stopping sets and codewords from perspectives of both linear algebra and graph theory. A method based on extrinsic message degree (EMD) is proposed

to construct irregular LDPC codes that have good cycle properties. As a result, these codes achieve near capacity capability and low error floors. The performance of different construction schemes are compared and the roles of cycles and stopping sets in affecting capacity and error-floors are analyzed. Next, a rate-compatible scheme based on LDPC codes is proposed. It is based on two techniques: parity puncturing and information nulling. Simulations show that this scheme achieves close-to-capacity performance over a wide range of code rates. Finally, a density evolution analysis is developed for compression of correlated sources using irregular LDPC codes. The standard density evolution algorithm is modified to incorporate the Hidden Markov Model (HMM) defining the correlation model between sources. The proposed algorithm achieves a compression rate close to the theoretical Slepian-Wolf limit.

Chapter 1

Introduction

Low-density parity-check (LDPC) codes were proposed by Gallager in the early 1960's [Gal62] [Gal63]. He defined an (n, d_v, d_c) LDPC code as a code of block length n in which each column of the parity check matrix contains d_v ones and each row contains d_c ones. Due to the regular structure (uniform column and row weight) of Gallager's codes, they are now called regular LDPC codes. Gallager also invented soft-decision and hard-decision iterative decoders based on message passing. Using hard-decision decoding, he showed simulation results for codes of block length around 500 bits. These results indicated that LDPC codes have very good potential for error correction. However, these codes were not long enough for the sphere packing bound to approach Shannon capacity, and the computational resources for longer random codes were decades away from being broadly accessible. For the ensuing three decades, LDPC codes received relatively little attention in the coding community.

Over the past two decades, a deeper understanding of the message passing algorithm defined on graphs has developed. Tanner [Tan81] introduced bipartite graphs to describe low-density codes and the sum-product algorithm based on these graphs. Wiberg *et al.* [Wib96] extended Tanner graphs by including state variables which

are invisible to decoders. Pearl [Pea88] systematically described the "belief propagation" algorithm operating on Bayesian networks. It has been recently shown that the forward/backward algorithm for turbo codes, the belief propagation algorithm for LDPC codes, and many other decoding algorithms for other graph-based codes, are variations of the generalized sum-product (S-P) algorithm operating on the so-called factor graphs (see [KFL01] [For01]). To avoid a confusion in notations, we will call all variations of the generalized S-P algorithms message passing.

In the mid-1990's, Berrou *et al.* [BGT] demonstrated the impressive capacity-approaching capability of turbo codes, which led to the explosion of interest in turbo codes and other long random linear codes. Turbo codes share many attributes with LDPC codes, mostly in the way message passing is performed during the iterative decoding process. These similarities stimulated a revived interest in previous work on LDPC codes. In 1999, MacKay *et al.* [Mac99] showed that LDPC codes have near capacity performance and proposed several empirical rules for constructing good random codes. Luby *et al.* [LMSS01] formally showed that properly constructed irregular LDPC codes can approach capacity more closely than regular ones. Richardson, Shokrollahi and Urbanke [RSU01] created a systematic method called density evolution to analyze and synthesize the *degree distribution* in asymptotically large random bipartite graphs under a wide range of channel realizations.

Recently, many interesting and equally important research topics have emerged aside from the fundamental topics mentioned above. As we know, the decoding complexity per bit for message passing depends on graph connectivity rather than block length, which makes decoding of very long blocks possible. However, the encoding complexity is quadratic in block length if dense generator matrices are used. [RU01] proposed an almost linear time systematic encoder that converts the parity-check matrix to "approximate" lower triangular form by permutation. The permutation

transform doesn't affect code performance because it conserves the sparsity of the original parity-check matrix. However, the transformed generator matrix has a special shape that allows it to encode most of the non-systematic bits recursively in linear time. Chung *et al.* [CRU01] proposed a Gaussian approximation approach to reduce the density evolution algorithm to a one-dimensional problem, with only a little loss in performance compared to the accurate density evolution. [MB01] showed that different scheduling in the message passing decoder gives different performance in the high SNR region. [DM98] showed by their simulations that LDPC codes over $GF(q)$ outperform binary LDPC codes. [Fos01] designed a reliability-based decoder to reduce the performance gap between message passing decoding and maximum likelihood (ML) decoding. [KLF01] designed LDPC codes based on finite geometries which have quasi-cyclic structures and very good minimum distance properties.

Density evolution determines the performance threshold for infinitely long codes whose associated bipartite graphs are assumed to follow a tree-like structure. Using the density evolution theory, Richardson *et al.* designed rate one-half LDPC codes achieving bit error rate (BER) 10^{-6} within less than one tenth of a dB from the capacity limit. However, the block length they used in order to achieve this performance was 10^6 bits which is too long for many applications. Bipartite graphs representing finite-length codes without singly connected nodes inevitably have many short cycles, which are neglected in the density evolution theory. Cycles in bipartite graphs compromise the optimality of the commonly practiced message passing decoding. If cycles exist, neighbors of a node are not conditionally independent in general, therefore graph separation is inaccurate and so is Pearl's polytree algorithm [Pea88] (which defines belief propagation as a special case).

Two finite-length analyses have been developed for LDPC codes recently: "stopping set analysis" [DPT$^+$02] for the binary erasure channel (BEC) and "projection

algebra and critical set analysis" [YSB]. The former predicts the performance of LDPC code ensembles with given degree distributions while the latter predicts the performance of a single LDPC code. Both analyses have to be improved because they have very high computation complexity for longer blocks (*e.g.*, several thousand bits).

Randomly realized finite-length irregular LDPC codes with block sizes on the order of 10^4 [RSU01] approach their density evolution *threshold* closely (within 0.8dB at BER $\approx 10^{-6}$) at rate 1/2, outperforming their regular counterparts [Mac99] by about 0.6dB. Most publications to date on this subject of irregular codes have focused on the performance relative to capacity, and do not consider performance at E_b/N_0 levels in the error floor region. In this paper, we repeated the irregular code construction method described in [RSU01] and extended their simulation to a higher SNR region. In the relatively unconditioned codes, an error floor was observed at BERs of slightly below 10^{-6}. In contrast, regular codes and almost regular codes ([KLF01]) usually enjoy very low error floors, apparently due to their more uniform Hamming distance between neighboring codewords and higher minimum distances.

MacKay *et al.* [MWD] first reported the tradeoff between the threshold SNR and the error floor BER for irregular LDPC codes versus regular LDPC codes. A similar tradeoff has been found for turbo codes ([BDMP98], [FW]). In the present contribution, we introduce code construction methods that specifically address the error floor issue. We present a design technique that requires all small cycles to have a minimum degree of connectivity with the rest of the graph. This technique lowers the error floors of irregular LDPC codes by several orders of magnitude with only a little cost in performance relative to capacity in the waterfall region of the BER versus E_b/N_0 curve.

The error floor of an LDPC code under maximum likelihood (ML) decoding depends on the d_{min} of the code and the multiplicity of d_{min} error events. However,

for randomly constructed codes, no algorithm is known to check if they have large minimum distances (This problem was proved to be NP-hard [Var97]).

As a result, the common approach has been to indirectly improve d_{min} through code conditioning techniques such as the removal of short cycles (girth conditioning [MB], [AEH]). Such conditioning is useful also because certain short cycles can cause poor performance in conjunction with iterative decoding even if they have a large d_{min} and would not be problematic for ML decoding.

Cycle properties play a critical role in determining error floors. We explore the relationship between cycles, stopping sets, and other attributes of the code and its associated bipartite graph. One argument we will make in this contribution is that not all cycles are equally problematic in practice. The more connected a cycle is to the rest of the graph, the less difficulty it poses to iterative decoding. A novel concept called "extrinsic message degree" (EMD) is introduced to help analyze the inner-structure of stopping sets. An efficient algorithm based on an approximation of EMD is proposed that constructs LDPC codes with good cycle properties and a correspondingly lower error-floor.

Many factors can change the characteristics (or states) of communication channels. These factors include multipath fading, temperature/moisture change, hostile jamming, and user-specified settings. A robust communication system should provide bandwidth close to capacity under various channel conditions. For example, the 3G CDMA specification IS-856 supports 12 data rates ranging from 38.4 to 2,457.6 Kbps. The corresponding code rate varies between 1/5 and 1/3. Traditionally, to achieve rate-compatibility, we have to independently build several subsystems that operate at different code rates, and switch between them according to channel state information. However, this scheme increases the design, setup and maintenance cost. A more efficient and flexible rate-compatible scheme is desired.

Punctured codes have long been used to achieve rate compatibility [Hag88] [WLS]. Compared to the traditional rate compatible codes based on convolutional codes and turbo codes, LDPC codes enjoy more freedom in puncturing pattern design, thus allowing for an almost continuous spectrum of code rates and more robustness to catastrophic events (stopping set puncturing). A density evolution algorithm was developed by Ha *et al.* [HM] to find asymptotically good puncturing profiles for LDPC codes.

We propose a rate-compatible scheme that combines parity puncturing and information nulling. Simulation results show that this scheme achieves close-to-capacity performance with low error floors across a wide range of code rates.

The last contribution is a density evolution analysis of a compression system for memory correlated binary sources using irregular LDPC codes as source codes.

It is well known that the problem of compressing correlated sources can be considered as a problem of channel coding with side information [Wyn74], [SVZ98]. The first approach to using practical channel codes in this context was presented in [PR]. More powerful turbo-like codes and iterative decoding schemes were introduced in [GF] and [GFZa]. Other work that utilizes turbo codes for source coding can be found in [BM], [AG]. Recent research [LXG02] has shown that the use of regular LDPC codes improves performance over turbo codes for the case of memoryless correlation. This result was extended in [GFZb] to the case of correlation with memory, where the memory is defined by HMMs.

The basic idea in [GFZb] is to incorporate the HMM in the graph that represents the code, and to apply the corresponding message passing algorithm over the whole graph. We extend this work to irregular LDPC codes. The standard density evolution algorithm is modified to incorporate the Hidden Markov Model (HMM) defining the correlation model between sources. Analysis and simulation shows that

the proposed irregular LDPC codes optimized with this algorithm outperform traditionally designed regular or irregular LDPC codes. The proposed algorithm achieves a compression rate close to the theoretical Slepian-Wolf limit.

Chapter 2 explores the relationship between several important graph structures: cycles, stopping sets, linearly dependent sets and edge expanding sets. A high-level description of their effects on the message passing algorithm is given. Furthermore, the sufficient and necessary condition for a set of variable nodes to be cycle-free is introduced.

Chapter 3 focuses on the inner-structure of stopping sets, especially how these sets are formed by clustering cycles. We will show that stopping sets have weak extrinsic message flow. To describe extrinsic message strength for stopping sets and variable node sets in general, we introduce EMD and its approximation ACE. A code construction algorithm based on ACE is given. The effectiveness and efficiency of this algorithm is discussed.

Chapter 4 describes the "smart" decoder and gives the simulation results for LDPC codes generated with the ACE algorithm. Different parameter schemes are compared.

Chapter 5 discusses a novel design of rate-compatible LDPC code that performs close to capacity over a large range of code rates. It combines three techniques: information nulling, parity puncturing, and lower triangular submatrix construction.

Chapter 6 develops a density evolution analysis of a compression system for memory correlated binary sources using irregular LDPC codes as source codes. In order to achieve this goal, the standard approach in density evolution is modified to incorporate the Hidden Markov Model (HMM) defining the correlation model between sources. The proposed scheme is then applied to the design of irregular LDPC codes that optimize the system performance.

Chapter 2

Analysis of Cycle Properties

This chapter will first introduce two equivalent descriptions of an LDPC code: the matrix and the bipartite graph. We then explore the relationship between several important graph structures, namely, cycles, stopping sets, codeword sets and edge expanding sets. The effect of these graph structures on code performance at high SNR will be discussed briefly. Furthermore, we will provide the sufficient and necessary condition for a set of variable nodes to be cycle-free.

2.1 Notation and Background Information

2.1.1 Representation of LDPC Codes

We wish to design an (n, k) binary systematic LDPC code where n is the block length and k is the number of information bits in one block. The code rate is $R = k/n$. The parity check matrix H is a full-rank $(n-k) \times n$ sparse matrix. The rows of H span the null space of the codeword space. H can be written as

$$H = \begin{bmatrix} H_1 & H_2 \end{bmatrix}, \tag{2.1}$$

where H_1 is an $(n-k) \times k$ matrix and H_2 is an $(n-k) \times (n-k)$ matrix. H_2 is constructed to be invertible, so by row transformation through left multiplication with H_2^{-1}, we obtain a systematic parity check matrix H_{sys} that is range equivalent to H

$$H_{sys} = H_2^{-1}H = \begin{bmatrix} H_2^{-1}H_1 & I_{n-k} \end{bmatrix}. \tag{2.2}$$

A systematic generator matrix can be obtained from H_{sys}

$$G_{sys} = \begin{bmatrix} I_k & \left(H_2^{-1}H_1\right)^T \end{bmatrix}. \tag{2.3}$$

The rows of G_{sys} span the codeword space. Obviously, $G_{sys}H^T = G_{sys}H_{sys}^T = 0$. It should be noted that although the original H matrix is sparse, neither H_{sys} nor G_{sys} is sparse in general. G_{sys} is used for encoding and the original sparse parity matrix H is used for iterative decoding.

An LDPC code can also be described by a bipartite graph. In this graph, n variable nodes form the left vertex set and $(n-k)$ constraint nodes form the right vertex set. In the case of a systematic code, the first k variable nodes are message nodes and the other $(n-k)$ variable nodes are parity nodes. If the entire set of variable nodes forms a valid codeword then the exclusive-or performed by each constraint node will be zero.

One column in the parity-check matrix corresponds to one variable in the bipartite graph. For convenience, we will use 'column' and 'variable' interchangeably in this paper.

The bipartite graphs and H matrices we are interested in have two further properties:

(1) There is at most one edge between any pair of nodes. In other words, there are no double-edges or other types of multiple-edges.

(2) There are no singly connected variable nodes, *i.e.*, the degree (number of neighbors) of any variable node is at least 2.

The first property ensures equivalence between the matrix and graph descriptions of the code. The second property will be used as part of a proof which relates graph structures known as stopping sets to cycles. As an example, the matrix and graph descriptions of a (9, 3) irregular code are shown in Fig. 2.1. Because a vertex cannot connect to another vertex from the same side in a bipartite graph, the length of any cycle in a bipartite graph is an even number. Fig. 2.1 also shows in solid lines a length-6 cycle that involves variable nodes v_0, v_4 and v_6.

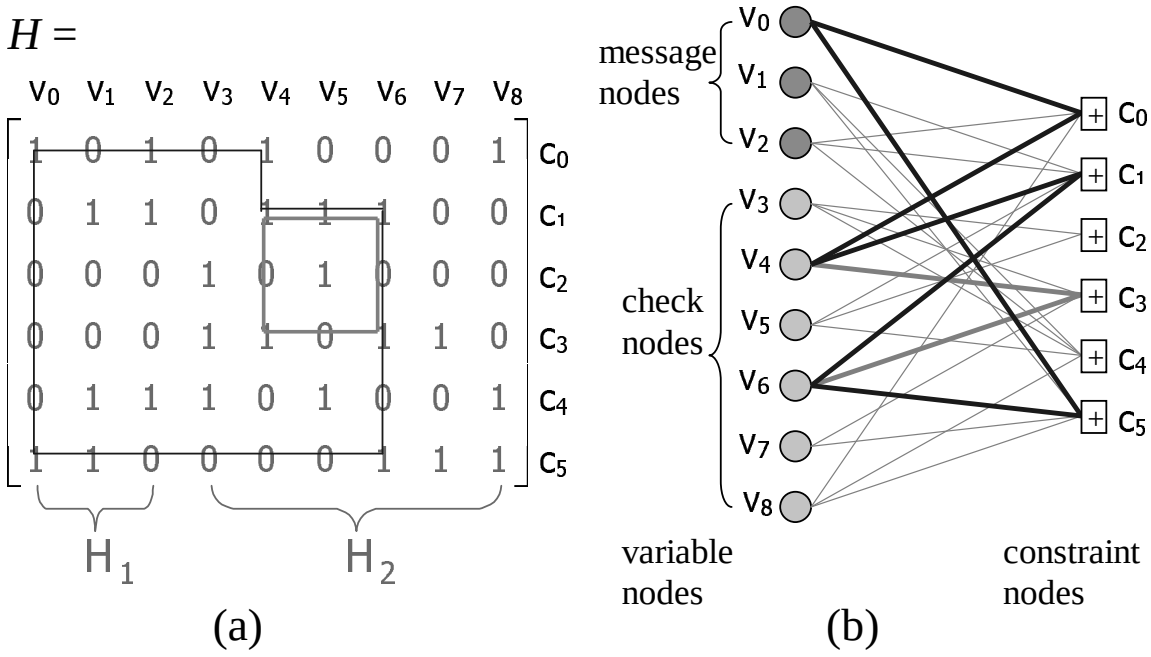


Figure 2.1: Matrix description and graph description of a (9, 3) code.

The systematic parity check matrix and the systematic generator matrix of this example can be derived according to Eq. 2.2 and 2.3. They are shown in Eq. 2.4

and 2.5

$$H_{sys} = \begin{bmatrix} H_2^{-1}H_1 & I_{n-k} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.4}$$

$$G_{sys} = \begin{bmatrix} I_k & \left(H_2^{-1}H_1\right)^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{2.5}$$

2.1.2 Message Passing Decoding

Let x be the transmitted signal corresponding to a variable bit with BPSK modulation. We define signal mapping in a way such that $x = -1$ if its corresponding variable bit is 1 and $x = 1$ if its corresponding variable bit is 0. Let y be the received signal which is equal to the sum of x and a noise. A message passing decoder tries to solve x's based on the knowledge about y's. There exist many ways to describe the soft-decision message passing algorithm (see [KFL01]). A very parsimonious and convenient one can be described as exchanging log-likelihood ratios (LLRs) between variable nodes and constraint nodes. Define u as an incident message to a variable node

$$u = \ln \frac{p(x = 1|y)}{p(x = -1|y)}, \tag{2.6}$$

and v as an emanating message from a variable node

$$v = \ln \frac{p(x' = 1|y')}{p(x' = -1|y')}, \tag{2.7}$$

where x' and y' have the same meaning as x and y except that they correspond to the variable that is located at the source of this message. As we can see, an important advantage of using LLRs is that probabilities such as 10^{-5} and $1 - 10^{-5}$ can be easily represented by $\ln \frac{10^{-5}}{1-10^{-5}} \approx -11.51$ and $\ln \frac{1-10^{-5}}{10^{-5}} \approx 11.51$. Fixed-point implementation is more accurate using LLRs than using probabilities because the finite-word effect is much reduced with LLRs.

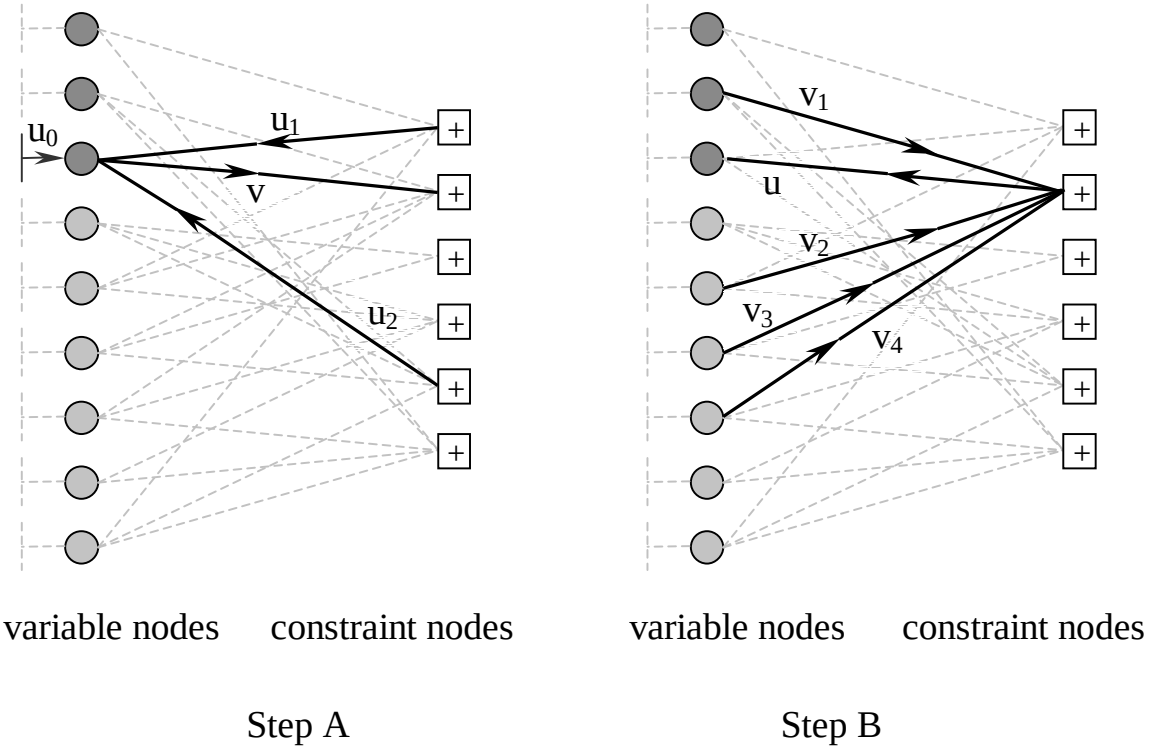


Figure 2.2: Illustration of one iteration in message passing decoding.

One iteration in message passing decoding consists of two steps. Step A, passing messages from constraint nodes to variable nodes; step B, passing messages from variable nodes to constraint nodes. These two steps are illustrated in Fig. 2.2. In step A, the half-edges attached to the variable nodes represent the *a priori* LLRs

determined by the received signal y and channel parameters. One message from a half-edge to its corresponding variable node is labeld u_0. The computation involved in these two steps can be written as Eq. 2.8 and 2.9 respectively (see [CRU01]).

$$v = \sum_{i=0}^{d_v-1} u_i, \tag{2.8}$$

$$\tanh \frac{u}{2} = \prod_{j=1}^{d_c-1} \tanh \frac{v_j}{2}, \tag{2.9}$$

where d_v and d_c are the degree of the corresponding variable node or constraint node respectively. Computation of Eq. 2.8 is simple because it only involves summation. However, Eq. 2.9 involves multiplication and hyperbolic tangent functions. We can prove that by defining

$$\begin{aligned} &\mathrm{f}(x) = -\ln\tanh\tfrac{x}{2} \approx \\ &\begin{cases} -\ln(1-2e^{-x}) = 2e^{-x} & if \quad x >> 1 \\ -\ln\frac{x}{2} & if \quad x \approx 0, \end{cases} \end{aligned} \tag{2.10}$$

Eq. 2.9 can be re-written as

$$\mathrm{sgn}(u) = \prod_{j=1}^{d_c-1} \mathrm{sgn}(v_j), \tag{2.11}$$

and

$$|u| = \mathrm{f}\left(\sum_{j=1}^{d_c-1} \mathrm{f}(|v_j|)\right). \tag{2.12}$$

where $\mathrm{sgn}(x)$ is the sign function. Eq. 2.12 is computationally efficient because the $\mathrm{f}(\cdot)$ function can be implemented by a look-up table and thus only summation is involved in calculating this equation. The curve for the $\mathrm{f}(\cdot)$ function is shown in Fig.

2.3.

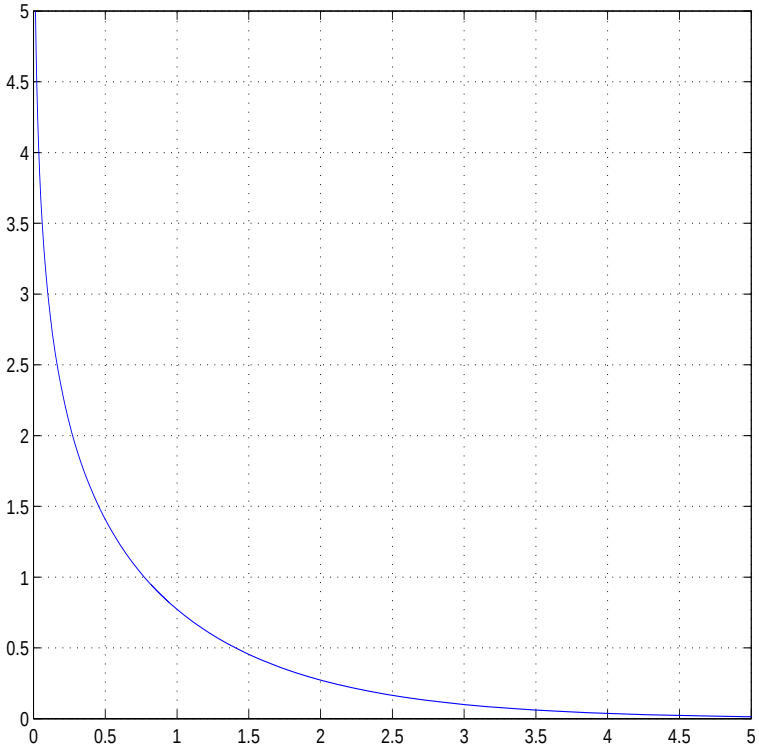


Figure 2.3: f(·) function used in message passing algorithm.

Due to the different nature of the functions involved in Eq. 2.8 and 2.9, u messages and v messages have different effect on the reliability of corresponding nodes. For example, in Eq. 2.8, if there are three input u messages

$$u_1 = -0.2, u_2 = 2, u_3 = -20, \tag{2.13}$$

then the output v message is

$$v = -0.2 + 2 - 20 = -18.2, \tag{2.14}$$

which has a value close to -20, the most reliable input message. In Eq. 2.9, if there are three input v messages

$$v_1 = -0.2, v_2 = 2, v_3 = -20, \tag{2.15}$$

then the output u message is

$$u = 2\tanh^{-1}(0.0997 \times 0.7616 \times 1.0000) = 0.152, \tag{2.16}$$

which is similar in magnitude to the lease reliable input message -0.2. This is because the hyperbolic tangent function saturates at high magnitude. The reliable messages barely affect the magnitude of the output u message because the hyperbolic tangent of them is close to either 1 or -1. Only the unreliable messages, whose magnitude lies in the linear region of the hyperbolic tangent function, reduces the reliability of the output u message. Particularly, in a BEC, an output v message will not be an erasure as long as there is an input u message that is not an erasure; an output u message will not be an erasure only if all of its input v messages are not erasures. To summarize this, we say:

The reliability of a v message is similar to that of the most reliable input u message; the reliability of a u message is at most that of the least reliable input v message.

We know that a BIAWGN channel can be quantized into a binary symmetric channel (BSC) in which all the LLRs have large magnitude at medium to high SNR. Note $\tanh(x/2) = sgn(x)$ for $|x| >> 1$. If we assume that all the u's and v's take value $\pm\infty$, Eq. 2.11 contains all the information about Eq. 2.9 because Eq. 2.12 is always satisfied. Note Eq. 2.11 is exactly the hard-decision parity check equation. Therefore, Eq. 2.8 can be viewed as the "soft" version of the parity check operation.

It should be noted that before message passing, all the unknown messages except *a priori* messages (denoted by u_0'sofTj 25.3-420(o3.79147 0 Td (view)d ('s.ould)Tj 36.6494 0 Td (b

$$\hat{x} = \text{sgn}\left(\sum_{i=0}^{d_v} u_i\right), \tag{2.17}$$

where $\hat{x}$ is the estimate of x and $sgn(\cdot)$ is the sign function. The difference between Eq. 2.17 and 2.8 is that the sum in Eq. 2.17 is taken over all the incident edges and half-edges, as opposed to only $d_v - 1$ edges.

Message passing decoding is optimal only for tree-like graphs. The performance of this decoding algorithm on a general (dense) graph is not guaranteed because of the wide existence of cycles. In contrast, sparse bipartite graphs are suitable for message passing decoding because they have better cycle properties than dense bipartite graphs. These properties will be analyzed in the next section.

2.2 The Relationship Between Cycles, Stopping Sets and Linearly Dependent Sets

Although the relationship between graph topology and code performance in the case of a specific code is not fully understood, work has been done to investigate the effects of graph structures such as cycles, stopping sets, codeword sets, and expanders. Here we give a specific analysis of how these four concepts are related. This analysis can be combined with density evolution to generate good irregular LDPC codes.

For brevity we denote cycle sets by C_d, stopping sets by S_d, codeword sets by W_d, and edge-expanding sets with parameter 1/2 by E_d. These sets, as well as the nature of the parameter d are described below. It is helpful to first illustrate the relationship between these sets (Fig. 2.4).

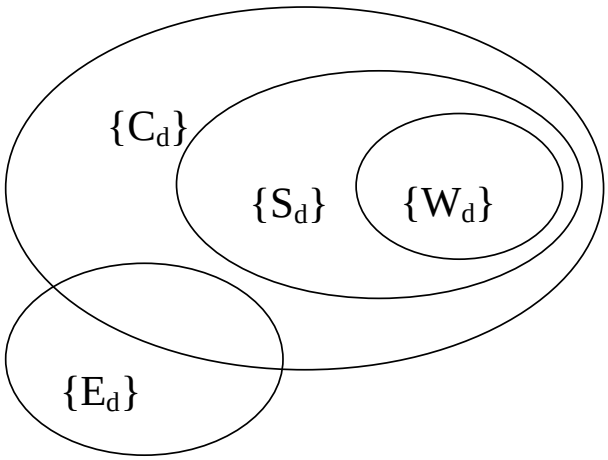


Figure 2.4: Relationship between C_d, S_d, W_d and E_d.

2.2.1 C_d Cycle Sets

Definition 2.2.1. (**Cycle**) A cycle of length $2d$ is a set of d variable nodes and d constraint nodes connected by edges such that a path exists that travels through every node in the set and connects each node to itself without traversing an edge twice.

Definition 2.2.2. (C_d **Cycle set**) A set of variable nodes in a bipartite graph is a C_d set if (1) it has d elements, and (2) one or more cycles are formed between this set and its neighboring constraint set. A set of d variable nodes does not form a C_d set only if no cycles exist between these variables and their constraint neighbors.

Note that the maximum cycle length that is possible in a C_d set is $2d$. Fig. 2.1 shows a length-6 cycle ($v_0 - c_0 - v_4 - c_1 - v_6 - c_5 - v_0$) and a length-4 cycle ($v_4 - c_1 - v_6 - c_3 - v_4$). Variable node set $\{v_0, v_4, v_6\}$ is a C_3 set. Variable node set $\{v_4, v_5, v_6\}$ is also a C_3 set although v_5 is not contained in the length-4 cycle. Di *et al.* defined a stopping set as follows, which we will show to contain cycles shortly.

A well known result is that message passing is accurate only for cycle-free graphs. We know that the number of edges in a tree equals to the number of nodes minus

one, thus a code whose graph is a tree has $n + (n - k) - 1 = 2n - k - 1$ edges. This number is too small for a practically useful finite length code. Actually the result given by density evolution says that all the variable nodes should be connected to at least two edges, which gives the minimal number of edges $2n > 2n - k - 1$. Therefore, cycles are almost inevitable and they make the message passing algorithm suboptimal. Furthermore, not all cycles are equally harmful. For example, it is well-known that short cycles among low-degree variable nodes represent severe code defects.

2.2.2 S_d Stopping Sets

Definition 2.2.3. (S_d **Stopping set [DPT+02]**) A variable node set is called an S_d set if it has d elements and all its neighbors are connected to it at least twice.

Variable node set $\{v_0, v_4, v_6\}$ in Fig. 2.1 is an S_3 set because all its neighbors c_0, c_1, c_3 and c_5 are connected to this set at least twice.

The following lemma shows that stopping sets always contain cycles. The effectiveness of message passing decoding on graphs with cycles depends primarily on how cycles are clustered to form stopping sets.

Lemma 2.2.4. *In a bipartite graph without singly connected variable nodes (such as one generated with a degree distribution given by density evolution), every stopping set contains cycles.*

Proof: A stopping set (variable nodes) and its neighbors (constraint nodes) form a bipartite graph where one can always leave a node on a different edge than used to enter that node. Traversing the resulting bipartite graph in this way indefinitely, one eventually visits a node twice, thus forming a cycle.□

Alternative proof: As explained before, if every node in an LDPC bipartite graph is at least doubly connected, the total number of edges is at least $2n$ which is larger than $2n - k - 1$, the number of edges required for the graph to be a tree.□

Lemma 2.2.5. *In a bipartite graph without singly connected variable nodes, stopping sets in general are comprised of multiple cycles. The only stopping sets formed by a single cycle are those that consist of all degree-2 variable nodes.*

Proof: A cycle that consists of all degree-2 variable nodes is a stopping set. To prove the lemma, we only need to show that if a cycle contains variable nodes of degree-3 or more, any stopping sets including this cycle are comprised of multiple cycles. Fig. 2.5(a) shows a cycle of arbitrary length $2d$ (here $2d = 8$ for demonstration). Assume that one variable node v_2 in this cycle has degree 3 or higher, v_2 must be connected to at least one constraint node out of this cycle (for instance c_1 in Fig. 2.5(a)). By the definition of a stopping set, c_1 must be connected to variable nodes in the stopping set at least twice. Therefore if c_1 is not connected to v_1, or v_3, or v_4, the stopping set must contain at least one more variable node (for instance v_5). The 'concatenation' of constraints and variables on to v_5 may occur across many nodes. However, to form a stopping set, eventually a new loop must be closed that connects the newest constraint in the chain to a variable on the chain or in the original cycle. Thus, the stopping set is comprised of at least two cycles.□

According to Lemma 2.2.5, the general view of stopping sets and cycles is given in Fig. 2.5(b). Two types of variable nodes comprise a stopping set. Variable nodes of the first type (denoted by solid curves) form cycles with other variable nodes; variable nodes of the second type (denoted by dashed curves) form binding structures that connect different cycles. It should be noted that both binding nodes and cycle nodes may have branches that lead to cycles containing variable nodes not in the current

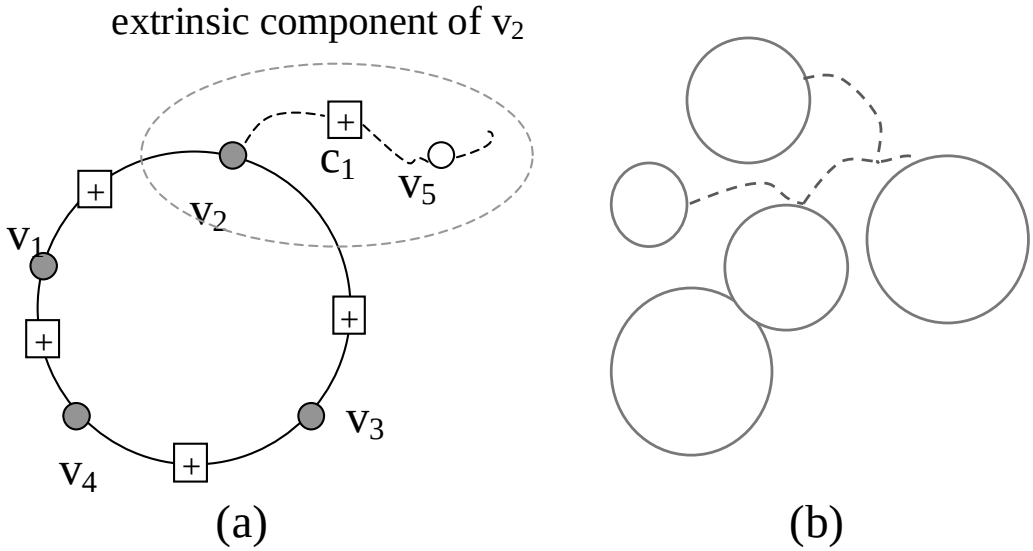


Figure 2.5: (a) Extrinsic concatenation (b) Structure of a stopping set

stopping set. Our proposed parity matrix design algorithm ensure that short cycles contain at least a given minimum number of 'extrinsic paths'. This leads to an increase in the minimum size of a stopping set.

Di, *et al.* [DPT+02] showed that in a binary erasure channel (BEC), the residual erasure bits after message passing iterative decoding are exactly equal to the maximum stopping set which is a subset of the originally erased bits. See the example in Fig. 2.6. Variable nodes v_0, v_4 and v_5 are erased. Any message passing from constraint nodes to these variable nodes, *e.g.*, the one from c_1 to v_6, is a function of the incident messages to c_1 from other variable nodes, which contain at least one erasure (the checked edge). Because an erasure is a bit that has equal probability to be 0 or 1, the resulting message remains erased and no further iterations can recover it. Therefore, stopping sets are the only type of "bad" cycles in a BEC. A natural conjecture is that stopping sets play an important role in other channels. In particular, consider the scenario where all members of a stopping set are received with poor reliability at the output of a binary-input additive white Gaussian (BIAWGN) chan-

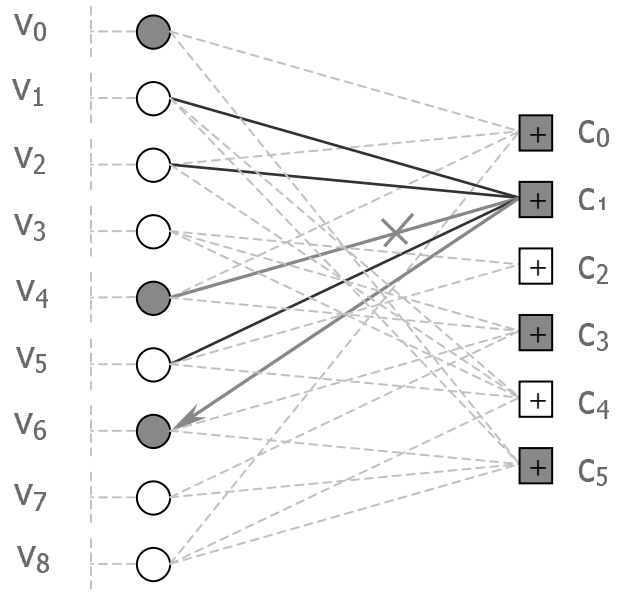


Figure 2.6: Erasure of stopping sets in BEC causes decoding failure.

nel. These reliabilities will be particularly slow to change in the course of message passing decoding as the reliability generated by any connected neighbor (constraint node) must also be low.

2.2.3 W_d Codeword Sets

Definition 2.2.6. (W_d **Codeword set**) A variable node set is called a W_d set if it is comprised of exactly d elements whose columns form a (weight-d) codeword.

Variable nodes set $\{v_0, v_4, v_6\}$ in Fig. 2.1 is the W_3 set corresponding to the codeword 100010100. A linear code with minimum distance d_{min} has at least one codeword with weight d_{min} and no non-zero codewords with smaller weight. Hence, there is at least one $W_{d_{min}}$ set but no W_d sets where $d < d_{min}$.

Erasing all the variables in a codeword set is the same as erasing all the non-zero positions of a binary codeword. Recovery from such an erasure is impossible even under ML decoding. Thus all codeword sets are stopping sets. This argument can

be formalized into the next theorem.

Theorem 2.2.7. *An W_d set must also be an S_d set.*

Proof: The binary sum of all columns corresponding to the variable nodes in W_d is the all-zero vector. Thus any neighbor (constraint node) of W_d is shared by the variable nodes that comprise W_d an even number of times, which means at least twice.□

In a linear block code, the lowest codeword (Hamming) weight is called the minimum distance d_{min}. Preventing small stopping sets also prevents small d_{min}. If a code has d_{min}, it must have an $S_{d_{min}}$ stopping set. Thus, avoiding all stopping sets S_d for $d \leq t$ ensures $d_{min} > t$.

However, small stopping sets do not necessarily represent low distance events. Indeed, an ML decoder can successfully decode an erased stopping set if a full column-rank sub-matrix is formed by the columns of the parity check matrix that are indexed by the stopping set variables. For example, $\{v_3, v_4, v_5, v_6, v_8\}$ in Fig. 2.1 is a stopping set that may be recovered by ML decoding (in the BEC case, simply solve a linear equation set). However, an erased stopping set can never be overcome by an iterative decoder.

With additive white Gaussian noise (AWGN), the magnitude of a corrupted signal can be so small that it can be effectively treated as an erasure. Hence the role of stopping sets can be translated to AWGN scenarios where variables with poor observation reliability are analogous to erasures. All stopping sets of small size are problematic. Some cause small distance, and all cause problems for iterative decoding. An obvious direction to take in order to generate codes well-suited to iterative decoding is to increase the size of minimum stopping set and reduce its multiplicity.

2.2.4 E_d Edge Expansion Sets

Edge expanders are known to have good minimum distance properties.

Definition 2.2.8. (**(α, γ) edge expander** [DPT$^+$02]) Let L be any subset of left nodes (variable nodes). Define $E(L)$ to be the number of edges connected to L and $N(L)$ to be the number of neighbors of L. An (α, γ) edge expander of an (n, k) code is a graph that has $N(L) > \gamma E(L)$ for all subsets with $E(L) \leq \alpha n$.

Methods that realize regular graphs with good edge expanding properties were proposed by Margulis in [Mar82]. However, a construction that can simultaneously satisfy a given edge expanding property as well as a given irregular degree distribution has yet to be proposed. We are interested in the special case of $(\alpha, 1/2)$ edge expanders.

Definition 2.2.9. (E_d **Edge-expanding set with parameter** $\gamma = 1/2$) A set of d variable nodes is called an E_d set if one-half of the number of edges emanating from it is less than the number of neighbors to which these edges connect.

The relationship between stopping sets and (α, 1/2) edge expanders is given by the following theorem.

Theorem 2.2.10. *(1) $\{S_d\}$ and $\{E_d\}$ are disjoint sets. i.e., An S_d set cannot be an E_d set and vice versa. (2) If all size-d subsets of variable nodes are $\{E_d\}$ sets then an $\{S_d\}$ set does not exist.*

Proof: According to the definition of E_d, neighbor nodes of E_d are connected to E_d less than twice on average. So there must exist at least one singly connected neighbor to E_d, this proves (1). It easily follows that if all size d subsets form E_d sets then no size-d subset may be an S_d set (2). □

For a code with minimal variable node degree 2, if all the subsets of an (n, k) code with d elements are E_d sets, where $d = 1, 2, ..., \text{ceil}\,(\alpha n/2)$, then this code is an $(\alpha, 1/2)$ edge expander. (This is a sufficient condition but not a necessary one, because an E_d set can have more than $2d$ edges if it contains variable nodes with degree higher than 2.) It follows that designing a code that avoids S_d sets is similar to the problem of generating as many E_d sets as possible. In the code design process that follows we improve the edge expansion property of short cycles, therefore indirectly reducing the occurrence of small stopping sets.

Fig. 2.4 outlines the relationships between the above-mentioned graph structures. Some examples help clarify these relationships. Fig. 2.7 (a) shows three columns that are not linearly dependent since their binary sum is not all-zero, hence, {0, 3, 5} is not a W_3 set. But because all neighbor nodes of {0, 3, 5} (corresponding to the first three rows) are connected to it at least twice, it is an S_3 set. Fig. 2.7 (b) shows one length-6 cycle in set {0, 3, 5} which makes this a C_3 set. However the last constraint node is singly connected to the variable node set and so this set does not form an S_3 set.

2.3 Cycle-free Sets

At this point, the value of removing small stopping sets is apparent. However, one might argue that simple girth conditioning accomplishes this because every stopping set contains cycles. The problem with traditional girth conditioning is that there are so many cycles. Fig. 2.8 illustrates a cycle in the support tree of variable node v_0 of Fig. 2.1. All the levels whose indices are odd numbers consist of constraint nodes and all the levels whose indices are even numbers consist of variable nodes. A cycle occurs if two positions in the support tree represent the same node in the bipartite

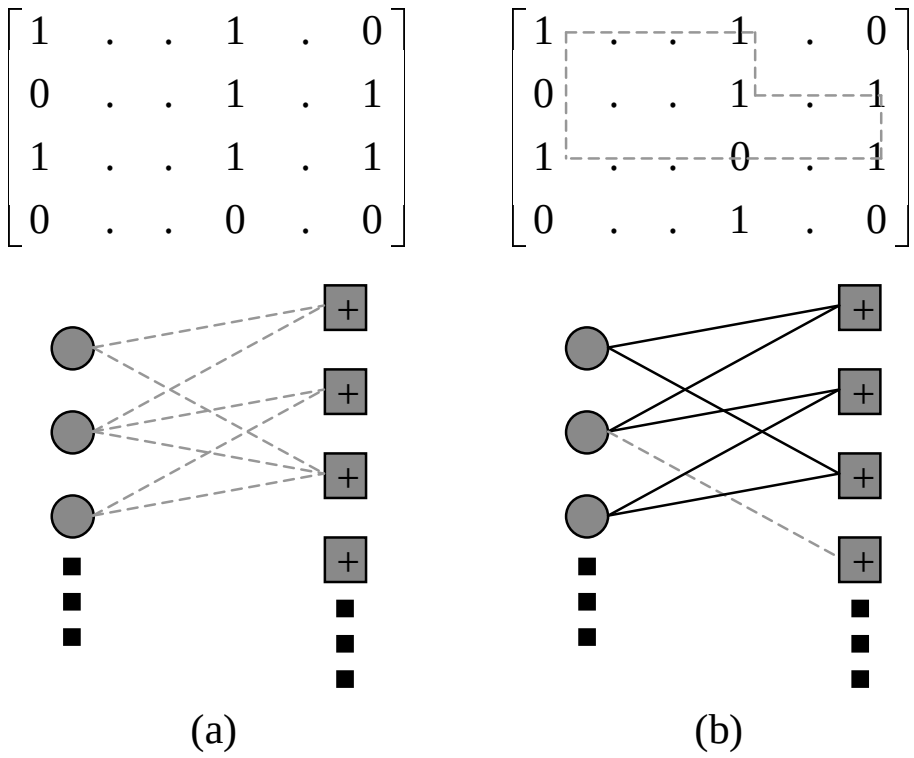


Figure 2.7: Examples (a)S_3 but not W_3 (b)C_3 but not S_3.

graph (*e.g.*, v_8 in level-3). To detect cycles of length up to $2d$ in the support tree of v_0, we need to expand its support tree d levels.

The number of nodes in the support tree grows exponentially with the number of levels expanded. To be short-cycle-free, all these nodes have to be different, so the longest cycle size we can avoid increases only logarithmically with block size (see [Gal62]). Since the logarithm is a slowly increasing function, girth conditioning of a finite length LDPC code is severely limited by block length.

Girth conditioning is especially problematic when there are high-degree nodes, as is common with degree distributions produced by density evolution. Recent girth conditioning techniques usually bypass high degree nodes. For example, in [MB], the edge-wise highest variable degree is only 3; in [AEH], the fraction of the highest degree variables λ_8 is only 0.025. As a result, girth conditioning was easier to

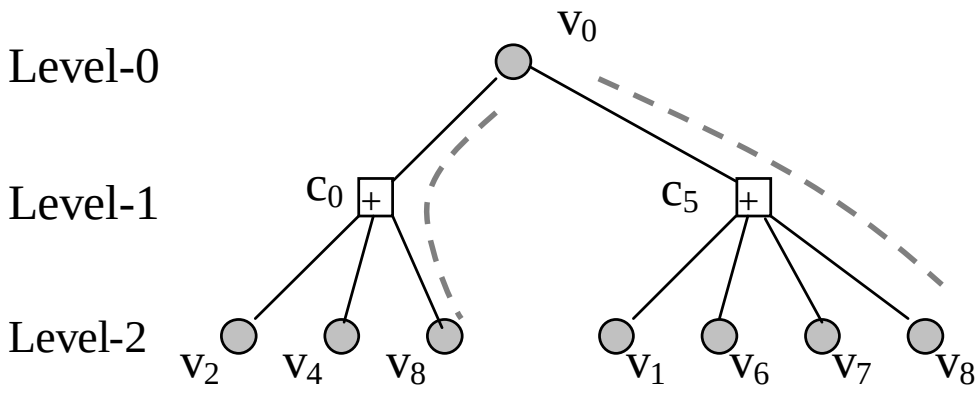


Figure 2.8: Traditional girth conditioning removes too many cycles.

perform. However, the capacity-approaching capability was sacrificed. High degree nodes are indicated by density evolution and lead to large stopping sets. The following arguments further discuss the cycle-related structures for high degree nodes and low degree nodes.

Definition 2.3.1. (**Cycle-free set**) A variable node set is called a cycle-free set if no cycle exists among its constituent variables.

Theorem 2.3.2. *A necessary and sufficient condition for a set of degree-*2 *variable nodes to be a cycle-free set is that this set is linearly independent.*

Proof: All sets that are not linearly independent contain codeword sets. Codeword sets are special stopping sets and stopping sets contain cycles (Lemma 2.2.4). For sufficiency, note that the constraint nodes taking part in a cycle among degree-2 nodes are each shared by exactly two variable nodes. Therefore the binary sum of columns (variables) taking place in the cycle is the all-zero vector and these columns are linearly dependent.□

Corollary 2.3.3. *A maximum of* $n-k-1$ *degree-2 columns of length* $n-k$ *may be linearly independent (cycle-free).*

Proof: Consider the $(n-k) \times (n-k-1)$ bi-diagonal matrix,

$$\underbrace{\begin{bmatrix} 1 & 0 & \cdots & & & 0 \\ 1 & 1 & & & & \\ 0 & 1 & \ddots & & & \\ \vdots & & \ddots & & & \\ & & & & 1 & 0 \\ & & & & 1 & 1 \\ 0 & \cdots & & & 0 & 1 \end{bmatrix}}_{n-k-1} \Bigg\}_{n-k}, \tag{2.18}$$

This matrix forms a rank $n-k-1$ basis of degree-2 columns each with dimension $n-k$. Any possible degree-2 column of dimension $n-k$ can be formed via a linear combination of columns in the above basis.□

Corollary 2.3.3 may also be considered a version of the Singleton bound where the restriction to degree-2 columns lowers the best possible d_{min} from $n-k$ to $n-k-1$.

Theorem 2.3.4. *In an (n, k) code free of degree-1 variables, a cycle-free variable node set* $\{v_1, v_2, ..., v_s\}$ *must satisfy* $\sum_{i=1}^{s} (d_i - 1) \leq n-k-1$*, where* d_i *is the degree of* v_i*.*

Proof: A degree-d variable node whose constraints are $\{c_1, c_2, ..., c_d\}$ can be conceptually replaced by a cluster of $d-1$ degree-2 nodes whose constraints are $\{c_1, c_2\}$, $\{c_2, c_3\}$, ..., $\{c_{d-1}, c_d\}$ respectively. As an example, Fig. 2.9 shows how variable node v_1 in Fig. 2.1 may be replaced by a cluster of two degree-2 nodes.

Because the indices of the constraints in a cluster are ordered, any cycle involving the degree-d node is equivalent to a cycle involving some of the degree-2 nodes in the cluster replacing the original node. Fig. 2.10 shows an example. Replace every variable in the set with its equivalent cluster. According to Corollary 2.3.3, at most

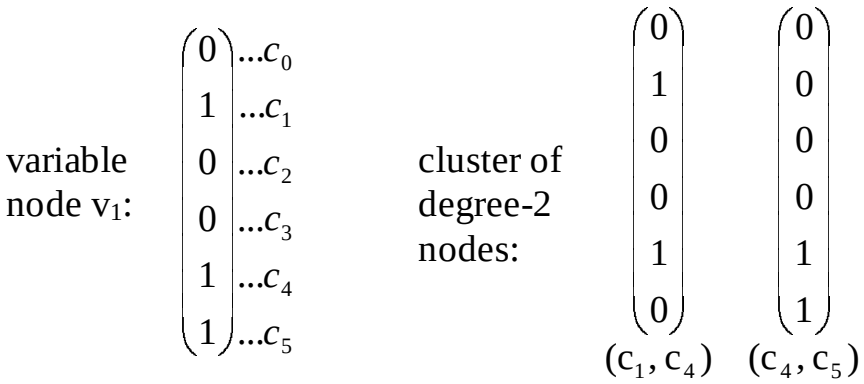


Figure 2.9: v_1 can be replaced by two degree-2 nodes.

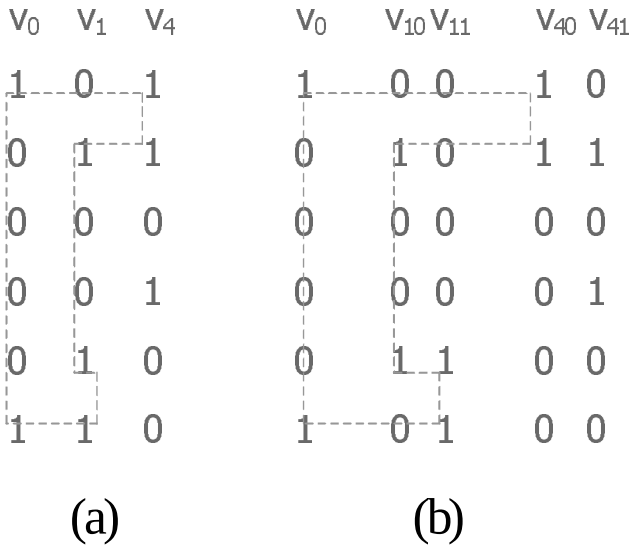


Figure 2.10: Replace v_1 by its cluster in a cycle.

$n-k-1$ of these resulting degree-2 nodes can form a cycle-free set. Thus the theorem is proved.□

Corollary 2.3.5. *In an (n, k) code free of degree-1 variables, no set of variable nodes whose cardinality is larger than $n-k-1$ can be cycle-free.*

Proof: Equality in the inequality of Theorem 2.3.4 is achieved with $n-k-1$ independent degree-2 variable nodes. Fewer variable nodes are allowed if some have higher degree.□

Lemma 2.2.5 and Theorem 2.3.2 show that for degree-2 variable nodes, cycles, stopping sets and codeword sets are equivalent. These structures are distinct for higher degree nodes following the Venn diagram of Fig. 2.4. Theorem 2.3.4 shows that higher degree nodes may b

Chapter 3

Extrinsic Message Degree and LDPC Code Design

The goal of LDPC code design is achieving good performance close to capacity while maintaining low error floors. These two goals are contradictory in general, as suggested by Urbanke in [DPT+02]:

> Steep cliffs are usually associated with high error floors. A large fraction of degree two left nodes leads to an initial rapid decline of the error probability. On the other hand a large degree two fraction usually causes problems towards the end of the decoding process.

This can also be explained intuitively in the following way. A low-error-floor code can dilute the effect of a few corrupted bits to a large number of bits at high SNR while a near-capacity code tends to localize errors at low SNR to prevent error propagation. Degree-two nodes widely exist in irregular LDPC codes with degree distribution generated by density evolution (see [RSU01]) whereas they do not exist in regular codes involving variable node of degree 3 or higher.

In this chapter, we will see that degree-two nodes allow very little extrinsic message flow, which is the major reason why randomly generated irregular codes have small stopping sets, and hence high error floors. Our goal in LDPC code design is to reduce error floor level, and maintain the capacity performance of irregular LDPC codes as well as possible.

For QPSK and BPSK modulation, Euclidean distance and Hamming distance are linearly related. Thus, it is reasonable to design codes for such modulations to focus on the Hamming distance spectrum. Minimum Hamming distance (d_{min}) is well-known to be related to the number of errors (t) that can be corrected reliably, $d_{min} = 2t + 1$. Minimum Hamming distance is also known to be linearly related to the number of erasures (u) that can be corrected, $d_{min} = u + 1$. We already know that the minimum stopping set size is equal to the smallest number of erasures that cannot be corrected by iterative decoding. Thus it is closely related to the Hamming distance (and hence Euclidean distance for QPSK and BPSK). Because the weight distribution of stopping sets is so closely related to the Euclidean distance spectrum, it is clear that LDPC designs focusing on the weight distribution of stopping sets is appropriate for AWGN channels as well as the binary erasure channel (BEC).

This goal can be achieved by ensuring that all stopping sets have at least some minimum number of variable nodes. However, no polynomial-time algorithm is known that removes small stopping sets explicitly, and our attempts to directly control stopping set sizes were too complex even to prevent very small stopping sets in a reasonable amount of time.

Lemma 2.2.5 shows that stopping sets are comprised of linked cycles. An efficient way to suppress small stopping sets is to improve the edge expanding properties of cycles in an irregular LDPC code. From the discussion of E_d sets, we know that constraint nodes singly connected to a variable node set provide good edge expansion

because these constraint nodes ensure useful message flows. Our algorithm achieves this by focusing on a parameter of variable node sets that we call the extrinsic message degree (EMD).

3.1 Extrinsic Message Degree

The essence of the message passing decoding algorithm is to find the probability of every variable bit based on the observation of all the other variable bits. From Chapter 2 we know that there are two types of messages passed in the bipartite graph: messages from constraint nodes to variable nodes (u messages) and messages from variable nodes to constraint nodes (v messages). These two types of messages have different reliability evolution rules. As shown in Eq. 2.8, the reliability of the output v message is similar to the reliability of the most reliable input u_i message; in Eq. 2.12, the reliability of the output u message is similar to the reliability of the least reliable input v_j message.

Consider the 'voting' example in Fig. 3.1. If the incident u message is positive, it "votes" variable node v_2 as $+1$, otherwise, it "votes" v_2 as -1. The magnitude of a message indicates its reliability. The output v message is generated as the sum of all incident u messages, which shows a compromise between "voters". If the magnitude of u is much larger than the magnitude of the messages from c_0 and c_4 to v_2, u will almost dominate the "vote" result at v_2. In contrast, the reliability generated by constraint node c_1 is not guaranteed solely by one v message. No matter how reliable this v message is, the reliability of c_1 also depends on other incident messages (from variable nodes v_1, v_4, v_5 and v_6). Flipping of any two incident v messages in Eq. 2.11 shows no detectable effects. In the case of a BEC where a stopping set (v_0, v_4 and v_6) is erased, at least two of the incident v messages to constraint node c_1 are

lost and any output u message c_1 generates will be erased no matter what incident messages it receives from variable nodes outside of this stopping set.

Thus we conclude: a non-erasure u message is useful in increasing the reliability of the corresponding variable node; a non-erasure v message is only useful in increasing the reliability generated by the corresponding constraint node if this constraint node is not a neighbor of an erased stopping set.

The difference between the effectiveness of u messages and v messages explains why the variable nodes of a good LDPC code can have very different degree whereas the constraint nodes typically have uniform degree. If some constraint nodes have very high degrees, these constraint nodes will become the "weak links" in the graph because the probability that two incident messages are corrupted for a high-degree constraint node is much higher than that for its low-degree counterpart. The degree difference of variable nodes does not have such an effect thus we can choose some variable nodes to have higher degree and make the decoding process converge faster.

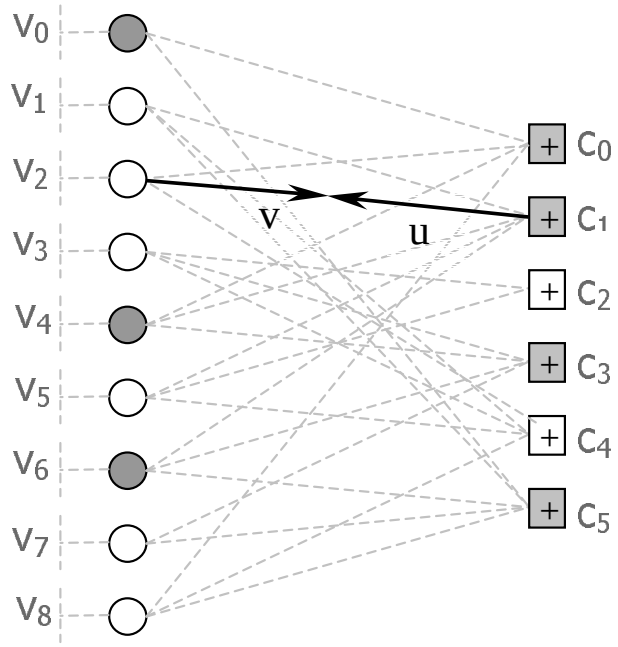


Figure 3.1: Message Effectiveness

Using the language of "extrinsic messages", the unreliability of an erased stopping set can be explained in the following way. A stopping set has no extrinsic u messages but only extrinsic v messages, which are not very effective in increasing the reliability of variable nodes in this set. Since all the neighbor nodes are connected to a stopping set twice or more, if all the variable nodes in the stopping set are erased, there will be no extrinsic messages that can recover them effectively. The next definition describes the effectiveness of the extrinsic message flow of a variable node set in general.

Definition 3.1.1. (**Extrinsic message degree**) An extrinsic constraint node of a variable node set is a constraint node that is singly connected to this set. The extrinsic message degree (EMD) of a variable node set is the number of extrinsic constraint nodes of this variable node set.

Obviously, the EMD is a metric representing the number of useful u messages incident to a variable node set. The EMD of a stopping set is zero. The only stopping set that contains a single cycle is a stopping set that consists of all degree-2 nodes. If this stopping set is erased, all of its u messages are erasures and all of the v messages are not useful in increasing the reliability generated by the neighbors of this stopping set.

Now we calculate the EMD of a cycle. If there are no variable nodes in a cycle that share common constraint nodes, the EMD of this cycle is equal to $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable in this set. Otherwise, there are constraint nodes connected to at least two variable nodes in a cycle, and the EMD of this cycle should be lower. In Fig. 3.2, variable nodes v_1 and v_3 are both connected to constraint node c_0. Thus the two edges that connect v_1, v_3 and c_0 are not extrinsic edges and the EMD of the large cycle is reduced by two. c_0 also breaks the large cycle into two smaller ones and generally speaking, constraint node sharing causes

short cycles.

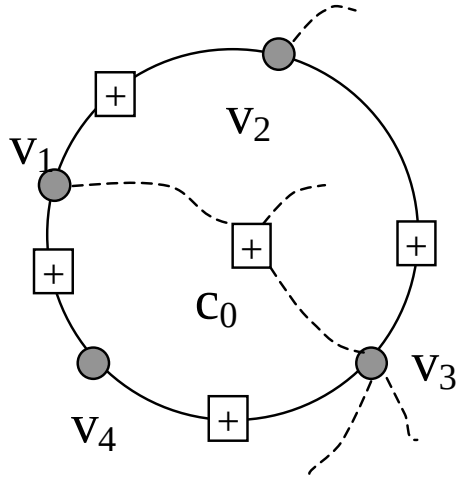


Figure 3.2: Sharing constraint nodes reduces the EMD of a cycle.

As was previously described, the high-SNR performance of an iteratively decoded LDPC code is limited by the size of the smallest stopping set (and its multiplicity, to be exact) in the code. The EMD of a stopping set is zero. A set of variable nodes with large EMD will require additional concatenation of nodes to become a stopping set. We will propose a conditioning algorithm that ensures all cycles less than a given length have an EMD greater than a given value. This technique statistically increases the smallest stopping set size. It also increases d_{min} because codeword sets are special cases of stopping sets.

3.2 Construction of LDPC Codes with Large Stopping Sets

3.2.1 Approximate EMD

First we consider the EMD of a generic cycle. If there are no variable nodes in a cycle that share common constraint nodes outside of the cycle, then the EMD of this cycle is $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable in this cycle. Otherwise, the EMD is reduced through constraint node sharing. To provide a calculable EMD metric, we neglect constraint node sharing and define an approximate cycle EMD.

Definition 3.2.1. (**Approximate cycle EMD (ACE)**) The ACE of a length $2d$ cycle is $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable in this cycle. We also say that the ACE of a degree-d variable node is $d-2$ and the ACE of any constraint node is 0.

ACE is an upper bound on EMD. The code conditioning algorithm to be proposed next is based on ACE instead of EMD. This approximation is reasonable since in this algorithm, *all* cycles shorter than a given length (including those formed through constraint node sharing) will be required to meet the ACE criteria. An LDPC code has property (d_{ACE}, η_{ACE}), if all the cycles whose length is $2d_{ACE}$ or less have ACE values of at least η_{ACE}.

3.2.2 ACE Algorithm Outline

The weight distribution of H is given by the density evolution algorithm [RSU01]. A typical distribution generated by this algorithm has one single concentration for constraint nodes (row weight) but has two separate concentrations around the highest

degree and the lowest degree for variable nodes (column weight). It is well known that variable nodes with more neighbors (constraints) experience lower decoder bit error rates. We assign higher degrees to the information bits to provide them with better protection. There is another advantage to this arrangement: cycles among degree-2 nodes are very harmful to code performance because their EMD is always zero and thus are always stopping sets. We note that if a code has $n-k-j$ degree-2 nodes ($j > 0$), it is possible to form a submatrix of degree-2 columns that has rank $n-k-j$ (see Corollary 2.3.3). Our codes are constructed such that the degree-2 variable nodes (columns) have this property when the density evolution distribution allows it.

In our codes, information bits come before parity bits (see Fig. 2.1). We assign column nodes such that degree decreases monotonically (*i.e.*, $d_i \geq d_j$ if $i < j$). Because high degree nodes converge faster, this arrangement provides more protection to information bits than to parity bits. The algorithm is as follows:

for $(i = n-1;\ i \geq 0;\ i--)$

begin

redo:

Randomly generate v_i according to *deg. distr.*;

if $i \geq k$ (*i.e.*, v_i is a parity bit)

begin

Gaussian Elimination (GE) on H_2;

if $v_i \in SPAN(v'_{i+1}, v'_{i+2}, ..., v'_{n-1})$

goto *redo*;

else

$v'_i \leftarrow$ the residue of v_i after GE;

end

ACE detection for v_i;

if $ACE < \eta_{ACE}$ for a cycle of length $2d_{ACE}$ or less

goto *redo*;

end

The Gaussian elimination process ultimately guarantees that the H matrix has full rank by ensuring that the $n-k$ columns of H_2 be linearly independent. For degree-2 variable nodes, independence entails freedom from cycles so that all degree-2 parity check nodes will be cycle-free. A caveat is that if Gaussian elimination is used in conjunction with a degree distribution that yields more than $n-k-1$ degree-2 nodes, then at least one of the $n-k$ parity check variables should have odd number degree (this can be achieved by column swapping). This follows immediately from Corollary 2.3.3.

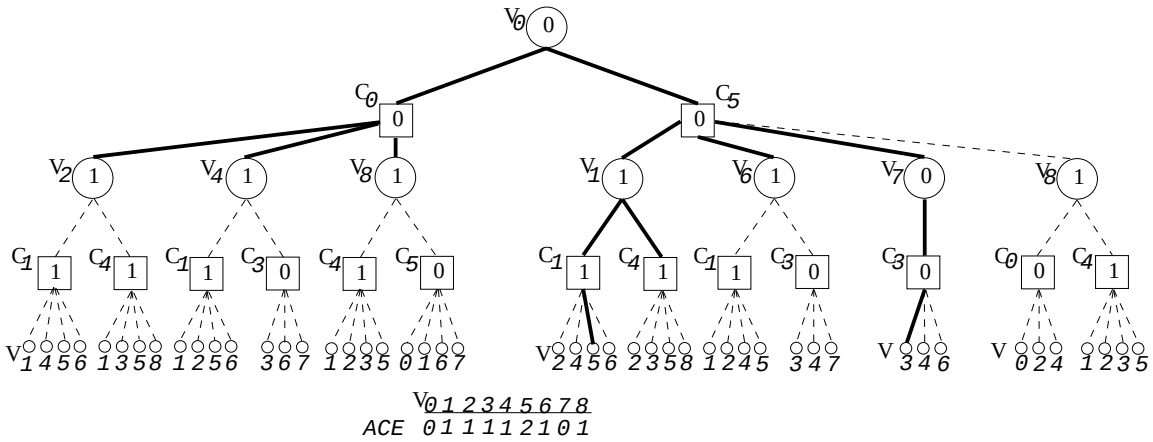


Figure 3.3: Illustration of an ACE search tree associated with v_0 in the example code of Fig.2.1. $\eta_{ACE} = 0$. Bold lines represent survivor paths. ACE values are indicated on the interior of circles (variables) or squares (constraints), except on the lowest level where they are instead described with a table.

3.2.3 ACE Detection in Tree Depiction

The ACE detection method can be equivalently depicted in two ways. The first one, based on support trees, is directly related to the graph structure. The second one, based on trellises, is oriented for algorithm implementation.

The tree depiction of ACE detection ($\eta_{ACE} = 0$) is given in Fig. 3.3. Here, variable and constraint node labels refer literally to those of the example code in Fig. 2.1 and the support tree that extends four levels below root node v_0 is portrayed. We define $p(\mu_t)$ to be the ACE of a path between root node v_0 and an arbitrary node μ_t (it can be either a variable node or a constraint node). Recall also that $ACE(\mu_t) = degree(\mu_t) - 2$ if μ_t is a variable, and $ACE(\mu_t) = 0$ if μ_t is a constraint.

ACE Detection of v_0

$p(\mu_t) \leftarrow \infty$ for all variables and constraints;

$p(v_0) \leftarrow ACE(v_0)$; Activate v_0 for level-0;

for $(l = 1;\ l \leq d_{ACE};\ l++)$

begin

  for any active node w_s in level-$(l-1)$

  begin

    Find its children set $Ch(w_s)$;

    for every child $\mu_t \in Ch(w_s)$

    begin

      $p_{temp} \leftarrow p(w_s) + ACE(\mu_t)$;

      if $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) < \eta_{ACE}$[1]

        exit with failure;

      elseif $p_{temp} \geq p(\mu_t)$

        Deactivate μ_t in level-l with respect to current parent w_s;

      else

        $p(\mu_t) \leftarrow p_{temp}$;

    end

  end

end

exit with success;

To explain the above algorithm, we need to recognize that a node should propagate descendants (be active) only if the path leading to this node has the lowest ACE value that any path to this node has had thus far. Therefore linear cost is achieved

[1]Note that this is the ACE of a cycle involving μ_t if $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) < \infty$.

instead of an exponential cost. Initially all the path ACEs can be set to ∞ (which means 'unvisited'). Note that cycles occur when a node is revisited, is simultaneously visited, or both. A *cycle ACE* equals the sum of the previously lowest path ACE to a node and the current path ACE to the node minus the doubly counted root and child ACE. When a cycle is formed by connecting two distinct paths from v_0 to μ_t we have cycle $ACE = p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)$, where p_{temp} and $p(\mu_t)$ are the ACEs of the two paths from v_0 to μ_t. Handling multiple simultaneous arrivals to the same node is a trivial extension where ACE minimization is performed sequentially across all arrivals.

In the example shown in Fig. 3.3, bold lines at each level describe the current set of *active paths*. In this example 'ties' are assigned the path whose parent has the lowest index. For instance the path (v_0-c_5-v_1-c_1) with $ACE = 1$ survives while, (v_0-c_5-v_6-c_1), (v_0-c_0-v_2-c_1), (v_0-c_0-v_4-c_1) each also having $ACE = 1$, perish. For an example of pruning occurring due to cycle detection on differing levels of the tree, observe that the path (v_0-c_0-v_8-c_5) with $ACE = 1$ does not survive since c_5 was visited at Level-1 and was accordingly assigned $ACE = 0$.

3.2.4 ACE Detection in Trellis Depiction

Fig. 3.4 provides a trellis depiction of the previous discussion (with two more stages added). A trellis instead of a full support tree is adequate for ACE detection because the ACE minimization is performed sequentially and only the minimum ACE needs to be stored. Again, a path ACE is stored for every variable node and every constraint node. An active path is a path that connects the root variable node and any other node with the lowest ACE value up to the current step. Active paths are marked by solid lines in Fig. 3.4. An *active node* is a node that connects to an active path at

the current step.

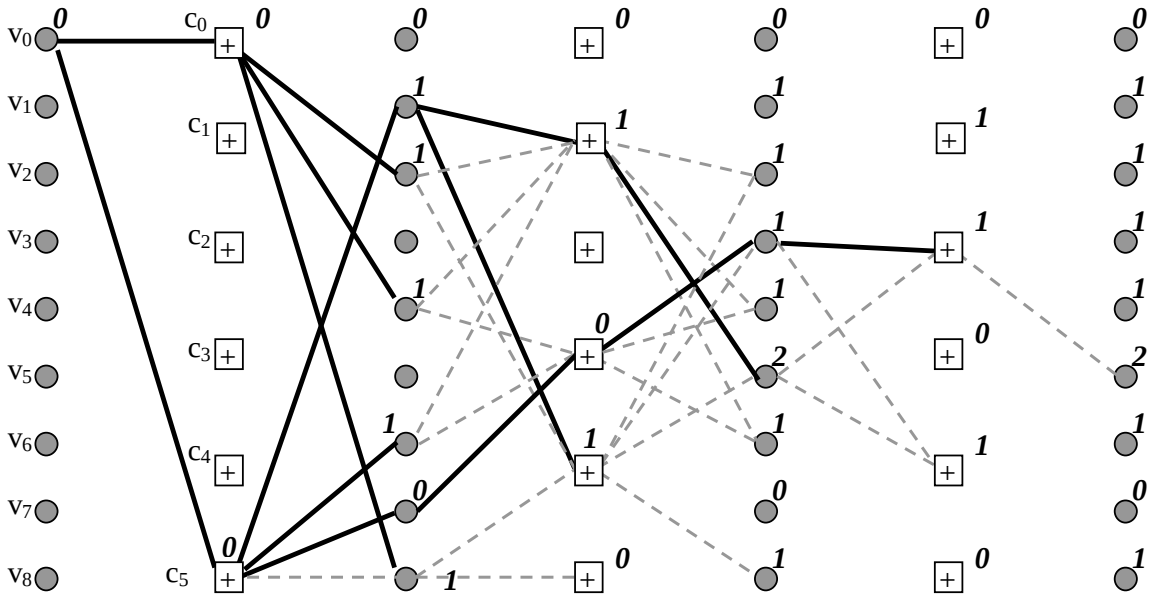


Figure 3.4: The Viterbi-like ACE algorithm. $\eta_{ACE} = 0$

Viterbi tree pruning yields a complexity at each root that is upper bounded by $d_{ACE} \times n \times (d-1)$ where d is the highest degree of any node in the graph, because the support tree is expanded d_{ACE} levels, for each level we have to consider at most n nodes, and every node has at most $d-1$ children. As shown in Fig. 3.4, relatively few nodes at a level are active, thus the actual computational burden is reasonable, even for block size on the order of 10^5 bits. The storage space needed is on the order of $n + (n-k)$ because only the current trellis level has to be saved. To further improve run-time, any active node with path ACE $p(\mu_t) \geq \eta_{ACE}$ can be deactivated because all the paths stemming from this node have an ACE value of at least η_{ACE}. As a special application of this argument, the user may generate columns for all variable nodes with degree $\eta_{ACE} + 2$ or higher without checking the ACE (since all descendants will have a path ACE of at least η_{ACE}).

Note that cycle detection can be implicitly performed in the above algorithm.

Unvisited variables are initialized to $p(\mu_t) = \infty$. When a variable is first visited, $p(\mu_t)$ is assigned a finite value. Upon subsequent visits to the same variable, ill-conditioned cycle detection is alarmed if the condition $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) \geq \eta_{ACE}$ is not satisfied.

other and the number of effective exits of these two cycles is only two.

A common attribute of these local solutions is that they create cycles shorter than the original ones. These short cycles are marked by dashed lines with arrows in Fig. 3.5. Since the length of these short cycles is usually much lower than $2d_{ACE} = 2 \times 10 = 20$, and these short cycles will all be conditioned by the ACE algorithm. Fig. 3.5 (e) shows a good solution of cycle exits when appropriate values of (d_{ACE}, η_{ACE}) are chosen. After joining two cycles, only one exit of each cycle is solved and four others are left for further clustering. Thus many more new variable nodes have to be introduced in order to form a stopping set. The clustering process finally ends in Fig. 3.5 (f), with four cycles in this stopping set, which obviously has a much larger size than the one in Fig. 3.5 (b). So far we have explained why this algorithm can avoid small stopping sets.

The complexity of checking if a newly generated variable node satisfies design rules is bounded by $2n - k$ because each node in a graph is stored once in the EMD table, which keeps the lowest EMD value of any paths between every node and the root variable node. Thus although the number of nodes in a support tree grows exponentially with the number of levels it is expanded, many nodes are redundant and the size of the EMD table is at most the number of nodes in the graph $n+(n-k) = 2n - k$.

Another question is whether "fake cycles" cause problems in this algorithm. As mentioned before, a "fake cycle" is a cycle in the support tree of a variable node whose two constituent paths merge before they are connected to the root variable node. In Fig. 3.6, v_1 and v_2 are found to be the same node, thus a cycle occurs. Because the two paths P_1 and P_2 merge at c_1 instead of v_0, this cycle is a "fake cycle" for v_0. The real cycle is C_1 (not including v_0). Since C_1 only contains old variable nodes (except v_0), it must satisfy design rules and thus its EMD value is at least η.

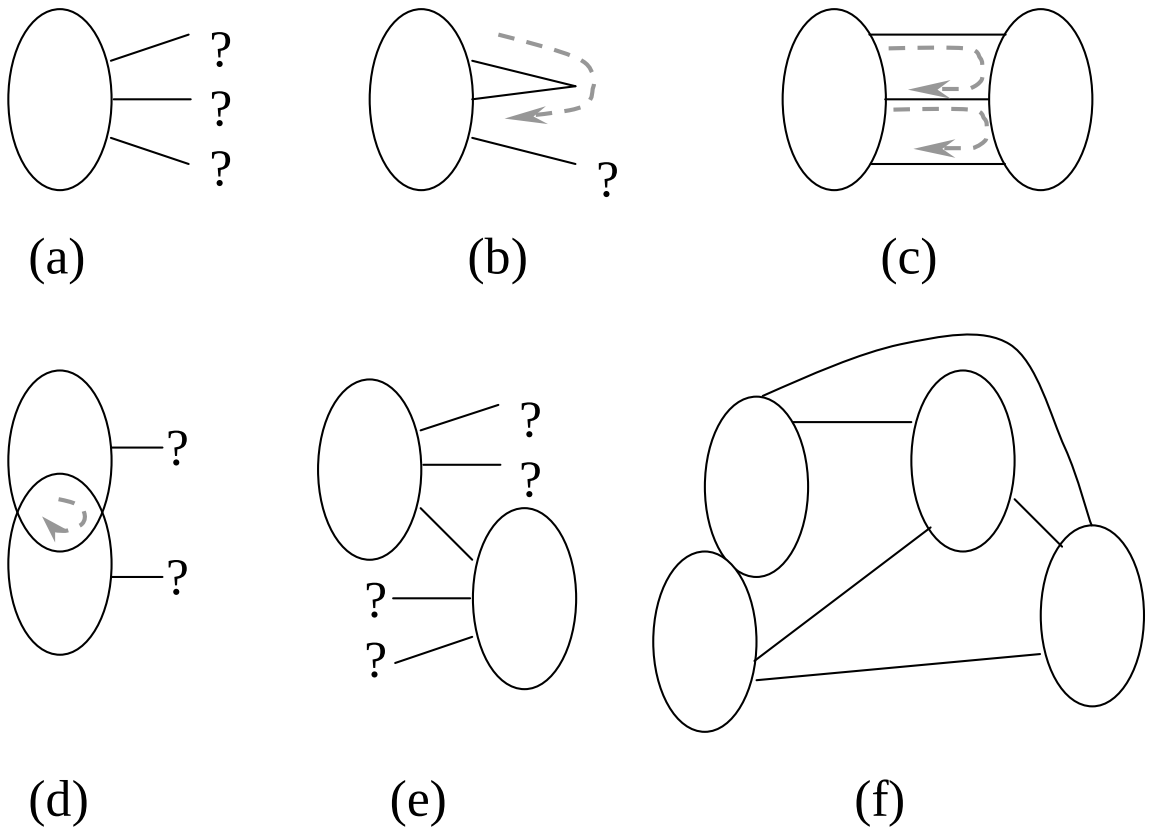


Figure 3.5: Cycle clustering.

Obviously the EMD value of the "fake cycle" is larger than or equal to that of C_1 because it contains more nodes. Thus a "fake cycle" automatically satisfies design rules if all the existing cycles obey the designing rules. Therefore, "fake cycles" will not be problematic.

The EMD algorithm can also be viewed as a "flooding" process, like those used to solve "routing" problems in the networking area. There are two types of nodes in this network: variable nodes and constraint nodes. The former introduces link "delay" which is equal to its EMD value; the latter introduces no link "delay". As we proceed, we keep the path from the root variable node to any node that can be reached in d_1 steps that has minimal cumulative "delay". When a cycle is formed, the cycle "delay" will be checked. This process goes on until all the nodes are checked

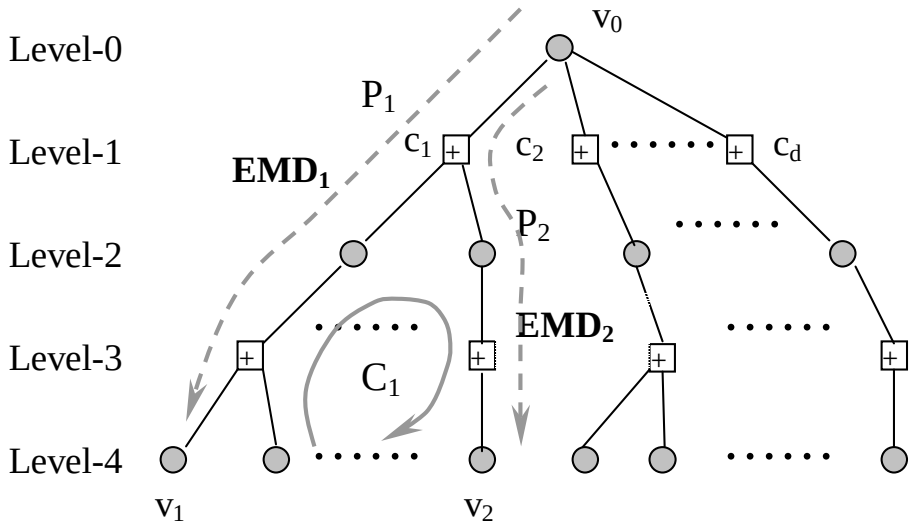


Figure 3.6: Fake cycles in the EMD algorithm.

and no violation of design rules are detected.

3.3 Summary

This chapter described the inner-structure of stopping sets and explained how they are formed by clustering cycles. An LDPC code design algorithm based on the approximate extrinsic message degree (ACE) was proposed. The simulation results will be shown in the next chapter.

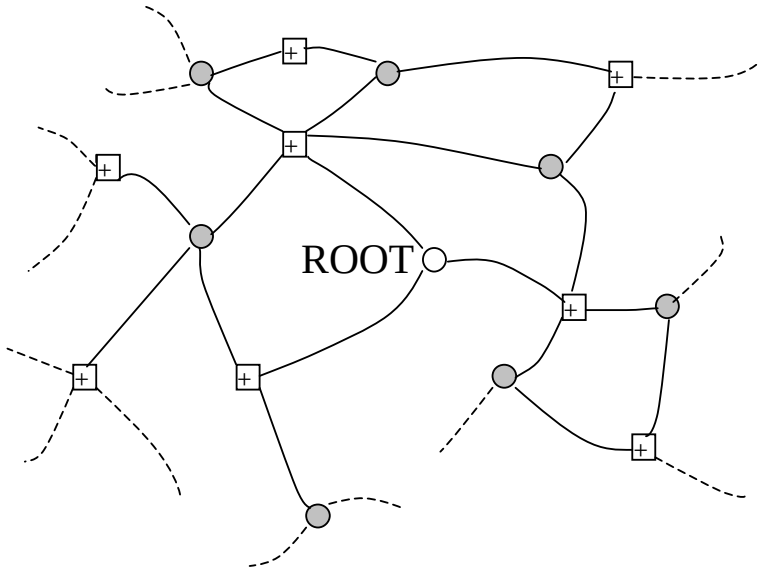


Figure 3.7: Our algorithm viewed as a flooding process.

Chapter 4

Simulation Results and Data Analysis

We present results for LDPC codes with three different block lengths.

4.1 Block-length 10,000 LDPC codes

We used the ACE algorithm to construct (10000, 5000) codes that have the irregular degree distribution given in [RSU01] with maximum variable node degree $d_v = 20$

$$\lambda(x) = 0.21991x + 0.23328x^2 + 0.02058x^3 + 0.08543x^5 + 0.06540x^6 + 0.04767x^7 + 0.01912x^8 + 0.08064x^{18} + 0.22798x^{19}, \tag{4.1}$$

$$\rho(x) = 0.64854x^7 + 0.34747x^8 + 0.00399x^9. \tag{4.2}$$

Any edge in a bipartite graph is connected to a left node (variable node) and a right node (constraint node). We refer to a left edge when we consider the degree

distribution related to variable nodes. Similarly we refer to a right edge when we consider the degree distribution of constraint nodes. $\lambda(x)$ describes the degree distribution of left edges. The coefficient of x^{i-1} in Eq. 4.1 represents the fraction of left edges that are connected to degree-i variable nodes. For example, the last term in Eq. 4.1 is $0.22798x^{19}$ which means 22.798% of edges are connected to degree-20 left nodes. The meaning of $\rho(x)$ is similar to $\lambda(x)$ except that it is defined for right edges and right nodes. The two edge degree distributions are shown in Fig. 4.1 and 4.2. The right edge degree distribution is very peaked whereas the left edge degree distribution has at least two separate concentrations. The low-degree concentration near degree 2 and the high-degree concentration near degree d_v are most prominent. For low rate codes and high d_v, less prominent concentrations may appear between the two major ones. $\lambda(x)$ and $\rho(x)$ are also subject to other constraints in order for the total number of left edges and that of right edges to be equal. (They are actually the same edges. "Left" and "right" are just names that indicates from what perspective we look at them.)

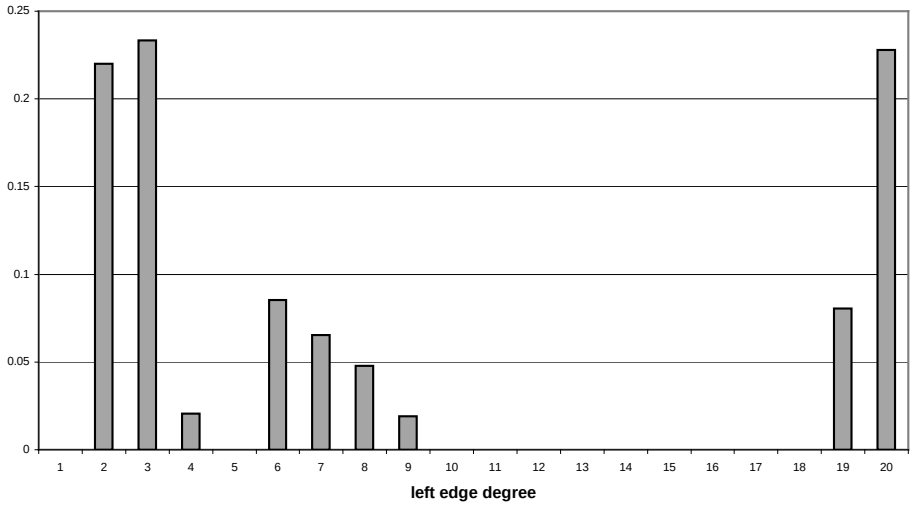


Figure 4.1: Left edge degree distribution.

When we design a real LDPC code based on the density evolution result, node-

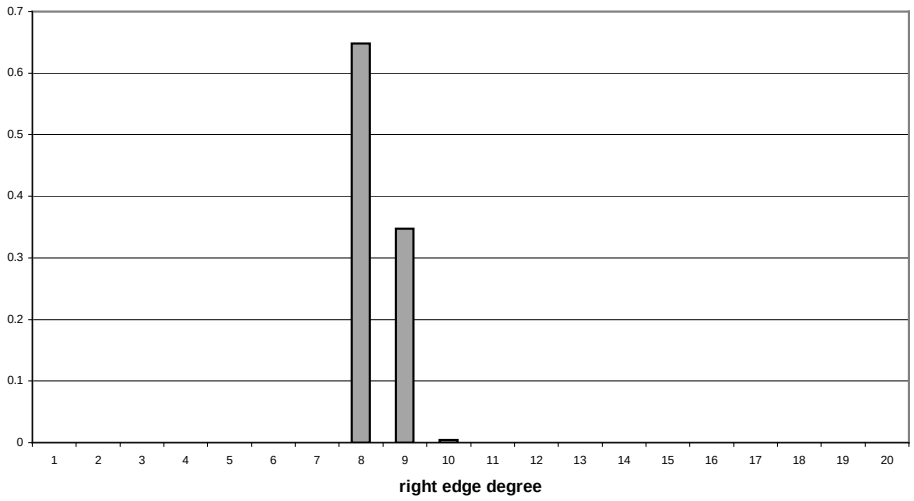


Figure 4.2: Right edge degree distribution.

wise degree distributions are more convenient than edge-wise degree distributions. They can be calculated as

$$\tilde{\lambda}_i = \frac{\lambda_i / i}{\sum_{j=2}^{d_v} \lambda_j / j}, \tag{4.3}$$

and

$$\tilde{\rho}_i = \frac{\rho_i / i}{\sum_{j=2}^{d_c} \rho_j / j}. \tag{4.4}$$

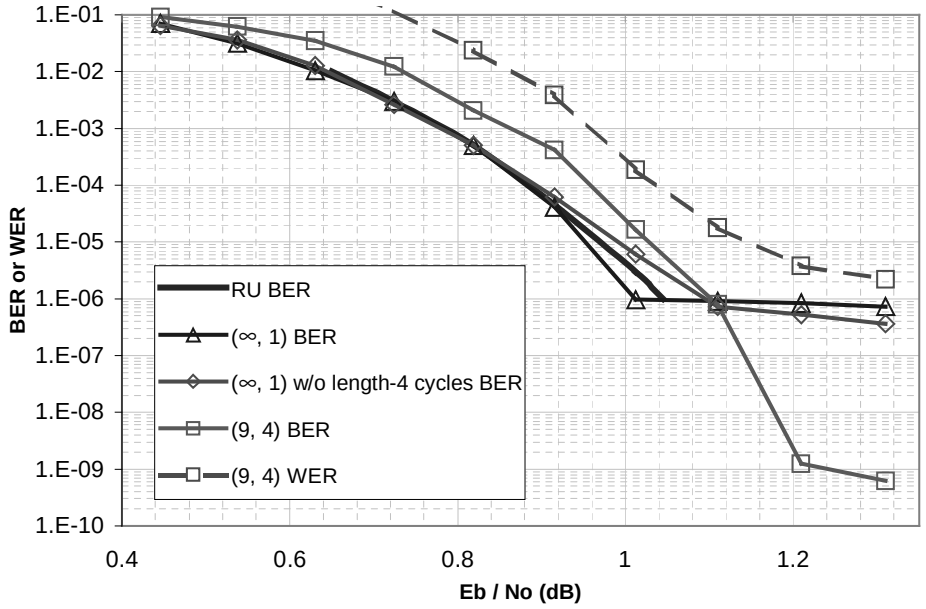
The node-wise degree distributions can also be easily translated to the edge-wise degree distributions.

$$\lambda_i = \frac{i\tilde{\lambda}_i}{\sum_{j=2}^{d_v} j\tilde{\lambda}_j}, \tag{4.5}$$

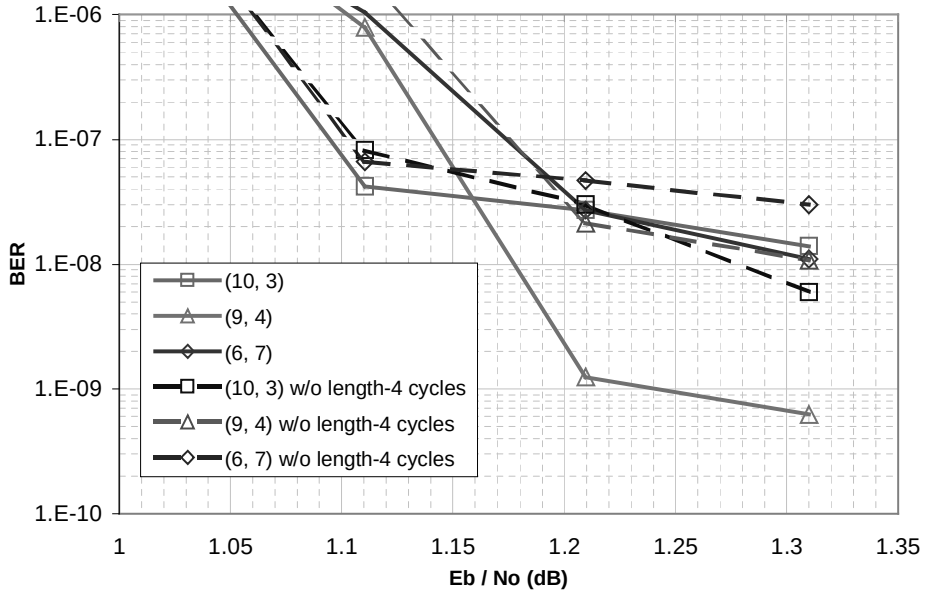
$$\rho_i = \frac{i\tilde{\rho}_i}{\sum_{j=2}^{d_c} j\tilde{\rho}_j}. \tag{4.6}$$

The encoded bits were sent through an BIAWGN channel. For each corrupted codeword, a maximum of 200 iterations were performed. Each simulation was stopped when 80 block errors were detected. The BER results and in one case the word (block) error rate (WER) results are plotted in Fig. 4.3.

A (d_{ACE}, η_{ACE}) code in Fig. 4.3 means that in this code, all cycles of length up



(a) hbt



(b) hbt

Figure 4.3: Results for (10000, 5000) codes.
The BPSK capacity bound at $R = 0.5$ is 0.188dB.

to $2d_{ACE}$ have ACE of at least η_{ACE}. Higher η_{ACE} values ensure better properties for the conditioned cycles and higher d_{ACE} values ensure that conditioning occurs for longer cycles. There is a tradeoff between d_{ACE} and η_{ACE}: increasing one parameter inevitably makes the other parameter more difficult to implement. For $d_{ACE} = 13, 10, 9, 7$, and 6, the highest η_{ACE} values that we could achieve were $2, 3, 4, 5$, and 7 respectively. If all degree-2 variables are parity bits (which is possible if there are fewer than $n - k$ of them), then the full-rankness of the parity matrix guarantees that no cycles will exist between degree-2 variables (see Theorem 2.3.2). In this case we can construct (∞, 1) codes since all cycles of any length must include nodes of at least degree-3 (and hence have ACE of at least 1). Such codes have the lowest level of conditioning that we consider and we use codes constructed in this manner to contrast the performance of stronger (d_{ACE}, η_{ACE}) conditioning levels.

As a benchmark, Richardson and Urbanke's 10^4-bit code [RSU01] (referred to here as the RU code) is included in Fig. 4.3(a). This RU code was constructed with the constraint that "all the degree-two nodes were made loop-free". Unfortunately, simulation results below BER 10^{-6} are unavailable for the RU code. The (∞, 1) code shown in our paper has a similar level of conditioning as the RU code and exhibit comparable performance. The (∞, 1) code ensures linear independence among parity bits which include all degree-two nodes and some degree-three nodes. The error-floor BER of the (∞, 1) code is around 10^{-6}. Pure length-4 cycle removal improves BER by half an order in magnitude over the (∞, 1) code at highest SNR, and adversely affects convergence. However, proper selection of d_{ACE} and η_{ACE} suppresses error floors significantly more. For example, the pure ACE conditioning code (9, 4) achieves approximately BER = 10^{-9}, three orders of magnitude below the (∞, 1) code.

Fig. 4.3(b) compares several ACE parameter sets with or without explicit length-4 cycle removal. Note that the lowest error floor was achieved at $d_{ACE} = 9$ and

$\eta_{ACE} = 4$ with no explicit removal of short cycles. As explained before, traditional girth conditioning treats all short cycles equally, thus making the removal of longer but still harmful cycles more difficult. On the contrary, the ACE algorithm effectively removes low-ACE cycles, while leaving shorter cycles intact, if they have a high ACE. By doing this, participation of high degree variables in cycles is encouraged. In fact, removing all length-4 cycles hinders the performance of the ACE algorithm.

We observe that there is a small penalty in the capacity-approaching capability of our low-error-floor codes. With more conditioning at this block size, the low-SNR performance of the code is slightly degraded, possibly due to a decrease in the randomness of the code structure. This tradeoff between error floor and low-SNR performance is a well-known characteristic of iteratively decoded codes ([MWD][BDMP98][FW]). Scheme (9, 4) is approximately 0.07dB away from the RU code at low SNR. However, even with this mild penalty these codes remain superior to regular codes in terms of their capacity-approaching performance. For example, we tested MacKay's (9972, 3, 6) regular LDPC code described in [MWD]. Although no error floors were detected for this code, it achieves BER $\approx 10^{-5}$ at $SNR \approx 1.7$dB, more than 0.6dB worse than our (9, 4) code. Thus the combination of density evolution optimized degree distributions and ACE construction achieves good performance over a wide SNR operating range.

To show the performance of our irregular codes compared to the RU code and MacKay's code, we plotted them in Fig. 4.4.

4.2 Shorter block lengths

To compare with other techniques at block lengths around 1000, we choose Mao's (1268, 456) code described in [MB] as a benchmark. Fig. 4.5 compares the perfor-

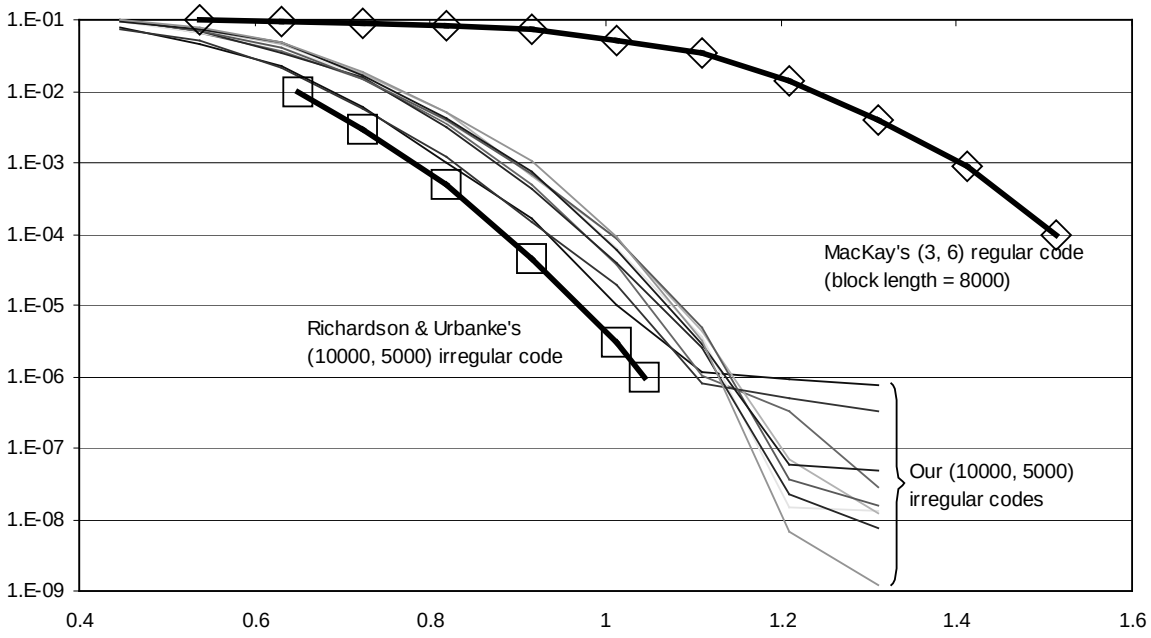


Figure 4.4: Comparison of code performance.

mance of the ACE-conditioned (1264, 456) code with that of Mao's (1268, 456) code (results drawn from [MB]).

It should be noted that degree distributions play an important role in conditioning schemes. With the low maximum variable degree ($d_v = 3$) distribution proposed in [MB], both the girth conditioning and the ACE conditioning are easy to perform. However, the ACE-conditioned code (∞, 3) is 0.2 dB better in BER at the high SNR region. If the density-evolution optimized distribution with $d_v = 14$ is imposed, the girth conditioning technique becomes more difficult due to the higher fraction of high degree nodes. However, the ACE algorithm still works well, outperforming Mao's code by 0.3dB, with no detectable error floors above $BER = 10^{-9}$.

We have also designed two ACE-conditioned (4000, 2000) codes to compare with Arnold's (4000, 2000)($d_v = 8$) code described in [AEH] (results drawn from [AEH], see Fig. 4.6). The degree distributions of the proposed code are Arnold's $d_v = 8$ degree distribution and the $d_v = 15$ degree distribution produced by density evolution.

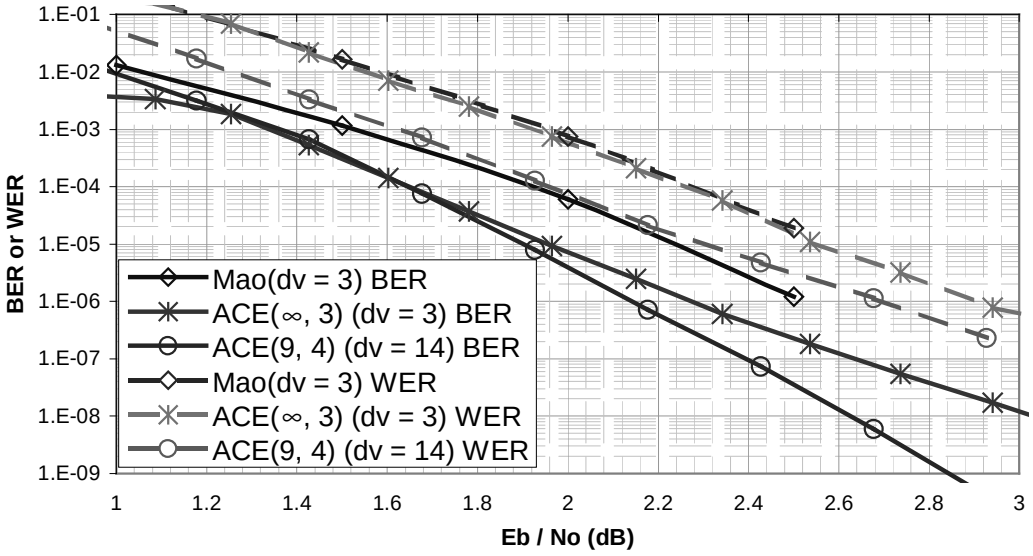


Figure 4.5: Results for (1264, 456) codes.
The BPSK capacity bound at $R = 0.36$ is -0.394dB.

As we can see from Fig. 4.6, by choosing Arnold's degree distribution, the ACE-conditioned code $(\infty, 6)$[1] achieves convergence 0.4dB worse than Arnold's code. This may result from the fact that the ACE algorithm is based on random generation and Arnold's progressive edge-growth technique successfully puts more structure in codes. We did not find error floors for the $(\infty, 6)$ code above BER $= 10^{-8}$. By choosing the density-evolution optimized distribution with $d_v = 15$, the ACE-conditioned code (9, 4) achieves a threshold SNR 0.1dB better than that of Arnold's code.

In summary, Arnold's code is better than the ACE-conditioned code for Arnold's $d_v = 8$ degree distribution, but that degree distribution is suboptimal. ACE conditioning of the density evolution degree distribution produces the best performance.

Fig. 4.7 shows the performance of some good turbo codes and our LDPC codes at block error rate 10^{-4}. Also shown is the Shannon sphere-packing bound [Sha59] [DDP98] for several code rates. This bound gives the best performance that a finite

[1] $d_{ACE} = \infty$ is achieved by extending the trellis until all the notes in the rightmost level are inactive

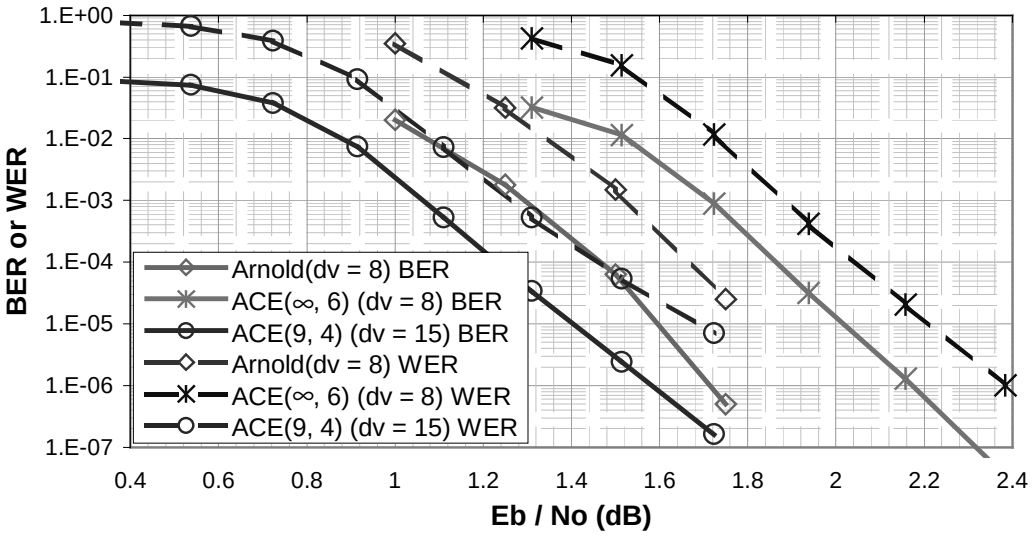


Figure 4.6: Results for (4000, 2000) codes.
The BPSK capacity bound at $R = 0.5$ is 0.188dB.

length code can possibly achieve. As we can see, both turbo codes and LDPC codes operate very close to the sphere-packing bound.

4.3 A Smart Decoder

We know in the BEC case, once a bit is recovered from erasure, it will not be erased again, so the number of bit errors is non-increasing with respect to the number of iterations. However, in the case of BIAWGN channels, we have observed a fluctuation of the number of bit errors in the medium SNR region (the transition between the error floor region and the waterfall region). A typical set of fluctuation curves at $SNR \approx 1$dB are shown in Fig. 4.8.

The number of information bit errors is denoted by the thin solid curve. The meaning of the other two curves will be explained shortly. These curves apparently show a periodic characteristic. The mechanism that governs the fluctuation is not inforwrmationHo

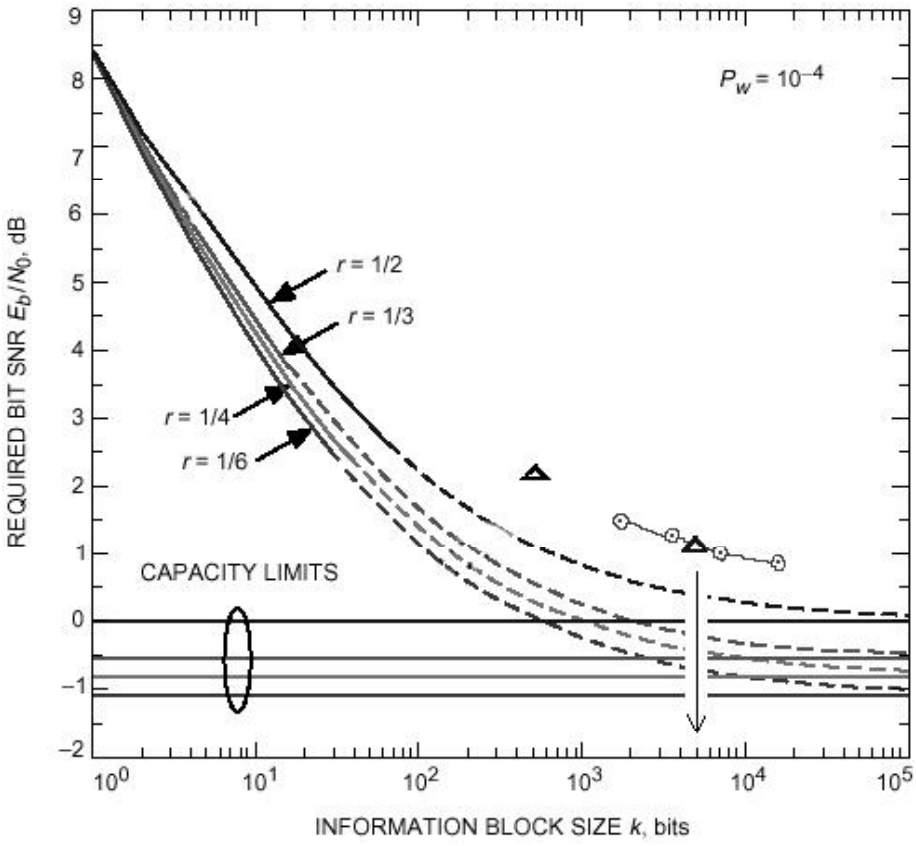


Figure 4.7: Sphere packing bound and some known good codes.
⊙: some good rate 1/2 turbo codes (data from JPL); △: rate 1/2 LDPC codes

severe BER fluctuation will occur. The amplitude of the oscillation decays slowly with time. In some cases, the errors are completely eliminated after a long time such as 1000 iterations. Therefore by allowing the decoder to run more iterations, we can improve code performance slightly but this will cause a long delay in the decoding process. However, simply by choosing a good decision point, we can severely reduce the number of iterations needed and achieve lower BER in the medium SNR region at the same time.

To choose a good decision point, we have to use an indicator. One candidate is the number of hard-decision constraints violated, which can be computed as the (Hamming) weight of the syndrome vector after each iteration

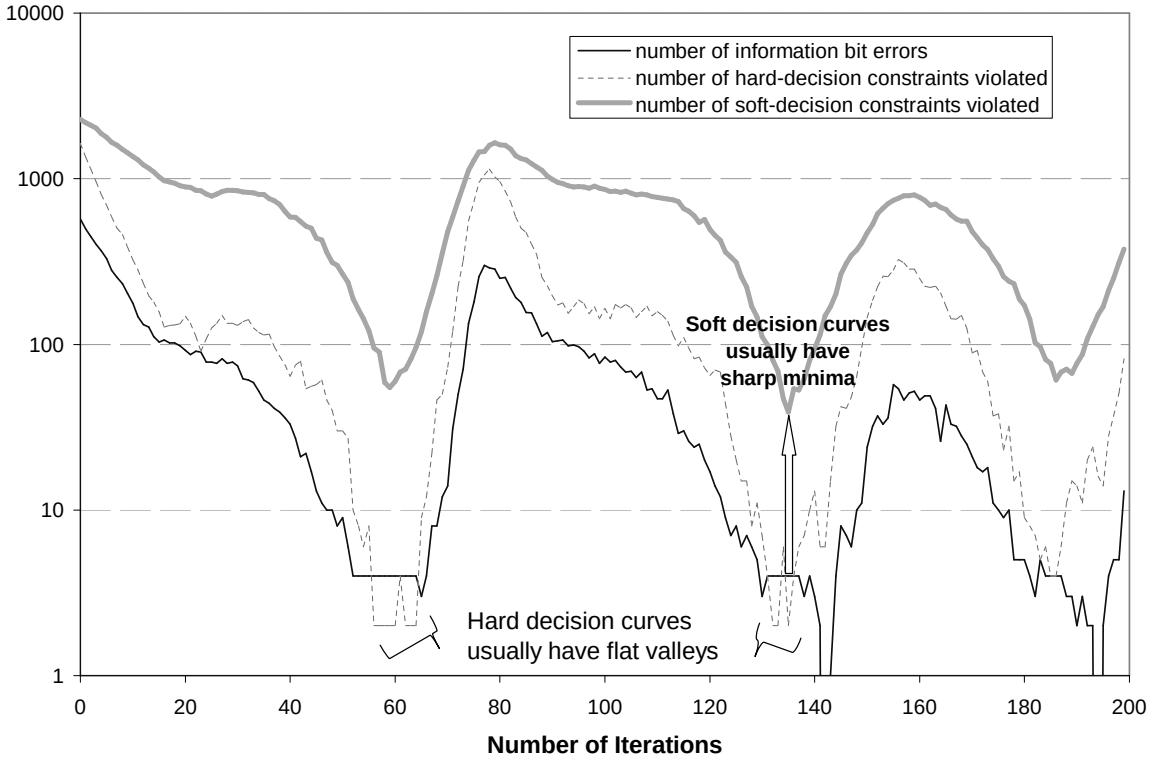


Figure 4.8: Fluctuation in number of bit errors in a BIAWGN channel.

$$s = rH^T = (c + e)\, H^T = eH^T, \tag{4.7}$$

where c is a codeword, e is the binary error pattern and r is the corrupted codeword.

From Eq. 4.7, we can see that the syndrome s is the binary sum of columns of H whose indices correspond to the positions of ones in the error pattern e. Because H is sparse and e has low weight at medium to high SNR, the positions of ones in these columns tend to be unique, thus the weight of s is approximately the weight of e times the average column weight of H. By selecting the point where the syndrome has minimal weight for our decision, we approximately minimize the weight of the error pattern as well. However, since this is a hard-decision indicator, it usually has flat valleys (see Fig. 4.8) which makes further choice of decision points within these flat regions difficult.

This problem can be solved by using the soft-decision "syndrome". We only have to make a slight change to Eq. 2.9 to obtain this soft-decision indicator.

$$\tanh \frac{s_{soft}}{2} = \prod_{j=1}^{d_c} \tanh \frac{v_j}{2}. \tag{4.8}$$

Note that the product is taken over all the d_c incident edges of a constraint nodes in Eq. 4.8. For a satisfied constraint node, the value of s_{soft} is a large positive number which forces the correspondent parity check to zero. Otherwise, for a violated constraint node, the value of s_{soft} is negative. Thus we can find the number of unreliable constraints by counting the number of negative soft syndromes among all the constraint nodes. This indicator is denoted in Fig. 4.8 by the thick solid curve. As we can see, it is a better indicator than its hard-decision counterpart since it has a sharper minimum region. In our simulations, soft decision iterative decoding is applied and a decision is made during the first 200 iterations when the smallest number of constraints are violated.

To make a fair comparison between our codes and other researchers' codes, we did not use the smart decision point algorithm in Fig. 4.3, 4.5, and 4.6. It should be noted that the smart algorithm can further reduce the error floor in those cases by half an order in magnitude.

4.4 Summary

This chapter showed simulation results for our LDPC code construction algorithm. We also introduced a "smart decision point algorithm" that chooses good decision points for BIAWGN channels. The next chapter will discuss future work on LDPC codes.

Chapter 5

Rate-Compatible LDPC Codes

Many factors can change the characteristics (or states) of communication channels. These factors include multipath fading, temperature/moisture change, hostile jamming, and user-specified settings. A robust communication system should provide bandwidth close to capacity under various channel conditions. For example, the 3G CDMA specification IS-856 supports 12 data rates ranging from 38.4 to 2,457.6 Kbps. The corresponding code rate varies between 1/5 and 1/3.

Rate-compatible coding is appropriate for communication systems that may experience a range of operating SNRs but that seek to adhere to a single underlying codec structure. Traditionally, to achieve rate-compatibility, we have to independently build several subsystems that operate at different code rates, and switch between them according to channel state information. However, this scheme increases the design, setup and maintenance cost. A more efficient and flexible rate-compatible scheme is desired.

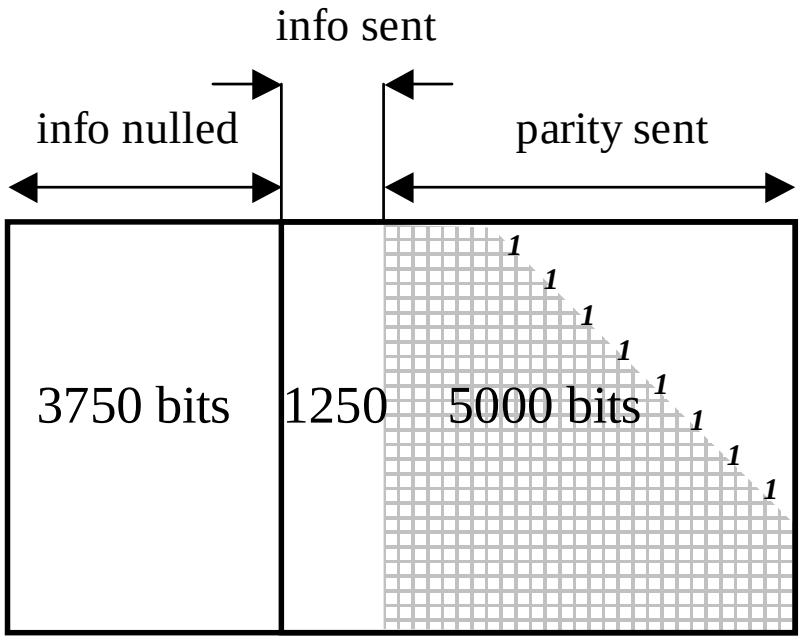
Punctured codes have long been used to achieve rate-compatibility. Mandelbaum [Man74] first proposed punctured Reed-Solomon codes with incremental redundancy. Cain *et al.* [CCG79] introduced punctured convolutional codes with code

rates $(n-1)/n$. This methodology was extended by Hagenauer [Hag88] to Rate-Compatible Punctured Convolutional (RCPC) codes, which allowed many continuous error protection level within a data frame. Soon after the invention of turbo codes [BGT] in the early 1990s, Barbulescu *et al.* [Bar95] developed the Rate-Compatible Punctured Turbo (RCPT) codes. Wesel [WLS] introduced punctured trellis codes, with emphasis on avoiding catastrophic events by proper selection of puncturing patterns. The puncturing pattern design was also discussed in [KS]. Ha *et al.* [HM] applied rate-compatible puncturing to the newly rediscovered low-density parity-check (LDPC) codes.

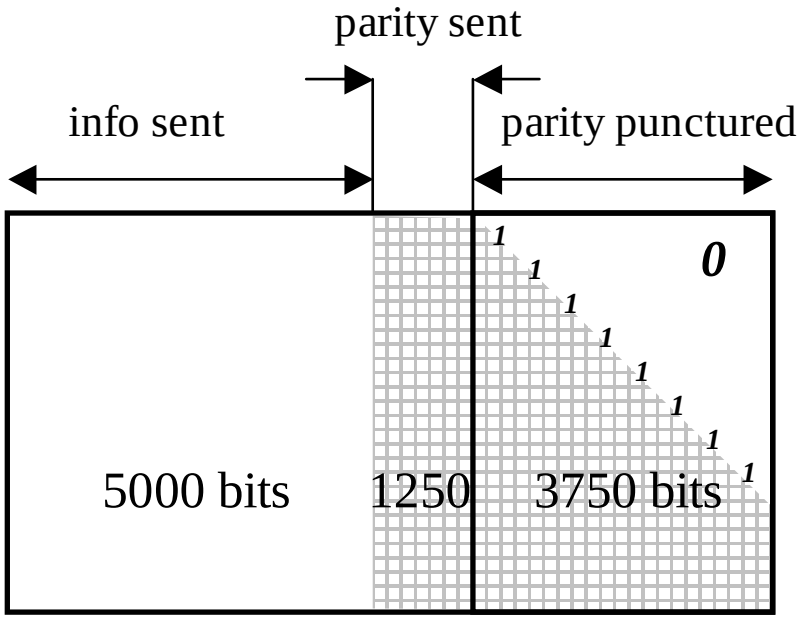
Compared to the family of convolutional codes and turbo codes (which include convolutional codes as constituent codes), LDPC codes can provide more flexible rate-compatible communications. For convolutional codes, punctured bits have to be distributed periodically with short period. In addition, to avoid catastrophic events, puncturing patterns have to be carefully chosen, which further limits the rate-variability. LDPC codes, on the other hand, are based on large random bipartite graphs, or equivalent matrices. Many well-established methods are available in graph theory and linear algebra to operate on LDPC codes, allowing for almost arbitrary code rates. Catastrophic events are alleviated by the intrinsic randomness of LDPC codes, and further by the technique to be proposed in this contribution.

5.1 Overview of the Proposed Rate-Compatible LDPC Codes

Fig. 5.1 shows an example of how the proposed method achieves low rate 0.2 and high rate 0.8 from a length-10^4 mother code that has rate $R_0 = 0.5$. Information bits



(a) Information nulling to achieve $R = 0.2$.



(b) Parity puncturing to achieve $R = 0.8$.

Figure 5.1: Proposed rate-compatible scheme for center rate $R_0 = 0.5$.

are on the left side (white area) and parity bits on the right side (shaded area).

To achieve $R = 0.2$ from the rate 0.5 mother code, zeros are used instead of payload data for the leftmost 3750 information bits in the encoding/decoding process. To achieve $R = 0.8$ from the rate 0.5 mother code, the rightmost 3750 parity bits are punctured. The number of information bits nulled and the number of parity bits punctured can be varied to achieve a wide range of code rates. For reasons to be described, rates above R_0 are achieved exclusively through parity puncturing and rates below R_0 exclusively through information nulling. We will propose a column degree assigning algorithm that endeavors to fit the degree sequence associated with a given code rate to the sequence that is asymptotically optimal for that rate.

5.2 Weight-Assigning Algorithm

We use density evolution ([RSU01][HM]) to find the node-wise degree distribution $\tilde{\lambda}_i^{(R)}$, $0 \leq R \leq 1$. To reduce design complexity, variable degree is only allowed to be 2, 3, 4, or 10. $\tilde{\lambda}_i^{(R)}$ is further normalized to

$$\tilde{L}_i^{(R)} = \begin{cases} \frac{1-R_0}{1-R}\tilde{\lambda}_i^{(R)}, & \text{if } 0 \leq R \leq R_0, \\ \tilde{\lambda}_i^{(R_0)}\left(1-\pi_i^{(R)}\right), & \text{if } R_0 < R \leq 1, \end{cases} \tag{5.1}$$

where $\tilde{L}_i$ represents the fraction of degree-i variables of the rate-R scheme with respect to the mother code and $\pi_i^{(R)}$ (see [HM]) represents the fraction of punctured degree-i variables. $\tilde{L}_i$ is plotted in Fig. 5.2.

The curves in Fig. 5.2 must be extrapolated to code rate 0 and 1 for a full

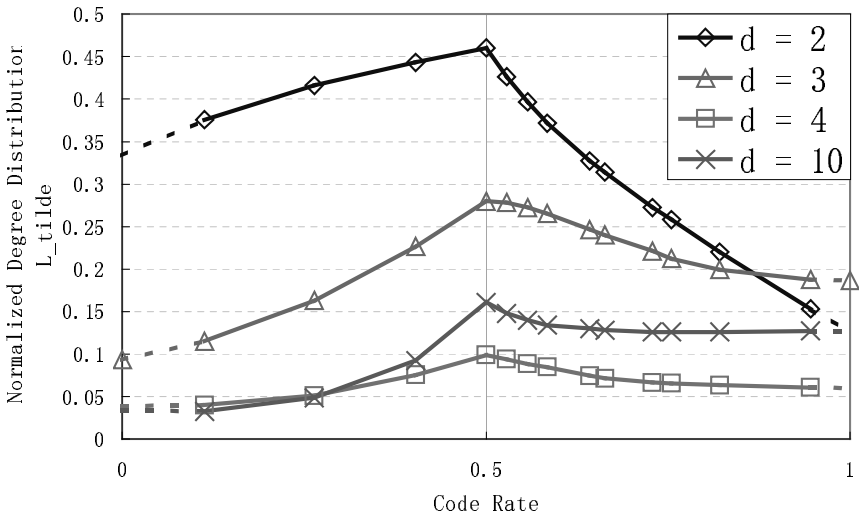


Figure 5.2: Normalized node-wise degree distribution $\tilde{L}_i$.

description of the mother code. The extrapolation must satisfied

$$\tilde{L}_i^{(0)} + \tilde{L}_i^{(1)} = \tilde{L}_i^{(R_0)}, \tag{5.2}$$

because $\tilde{L}_i^{(0)}$ describes the normalized distribution of the parity portion of H, $\tilde{L}_i^{(1)}$ describes the normalized distribution of the information portion of H, and their sum

Column Degree Allocation

$n_i = 0,\ i = 2, 3, ..., d_v - 1;$

for (column $\eta = 1; \eta \leq n; \eta{+}{+}$)

$x = \frac{\eta}{n}$;

if $(x < R_0)$

$p_i = n \times \left\{ \tilde{L}_i^{(R_0)} - \tilde{L}_i^{(R_0 - x)} \right\} - n_i,\ i = 2, 3, ..., d_v - 1;$

else

$p_i = n \times \tilde{L}_i^{(1 - x + R_0)} - n_i,\ i = 2, 3, ..., d_v - 1;$

endif

$j = \arg\max_i \{p_i\}$;

Assign the degree of column η to j;

$n_j{+}{+}$;

end

5.3 Advantage of the Proposed Scheme

We propose the lower triangular structure in Fig. 5.1 for two reasons.

5.3.1 Efficient Encoding

First, the parity matrix satisfies the structure proposed by [RU01] and hence has an almost linear time encoder.

A traditional LDPC encoder involves multiplying the message vector and the dense generator matrix. The computation complexity of this algorithm is $O(n^2)$. Richardson and Urbanke [RU01] proposed an encoder with a lower triangular structure similar to Fig. 5.1. Their approach can be described as:

1. Construct a random parity matrix

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}, \tag{5.3}$$

where T is a lower-triangular matrix. A and C correspond to information bits and all other submatrix form a square matrix corresponding to parity bits. A codeword is also divided into three parts: $c = (m\ p_1\ p_2)$, where m represents the message (information bits), p_1 and p_2 represent the parity bits.

2. Obtain an alternative parity matrix by row-transform

$$H = \begin{pmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{pmatrix}. \tag{5.4}$$

3. Define $\Phi = -ET^{-1}B+D$. If Φ is non-singular, we can calculate $p_1^T = -\Phi^{-1}(-ET^{-1}A+C)m^T$ in almost linear time, and calculate $p_2^T = -T^{-1}(As^T + Bp_1^T)$ in linear time with back-substitution.

The proposed algorithm performs puncturing on p_2. Our approach is different from the one described in [RU01] in two ways. First, we construct the matrix in a column-by-column fashion instead of by column and row permutation. The size of the upper triangular matrix is adjustable as long as it covers all the punctured bits in the highest rate scheme. Second, in our algorithm, Φ is guaranteed to be non-singular by the full-rankness of H.

5.3.2 Error Floor Suppression

Second, the proposed structure can suppress error floors. Recall that to form a stopping set, each constraint neighbor of a variable set must connect to this variable set at least twice. Any column (variable) subset of portion (2) in Fig. 5.1(b) is not a stopping set,

because the leftmost column of this subset has at least a neighbor (the upmost "1" of this column) that is singly connected to this set.

5.3.3 Why Nulling and Puncturing?

Alternatively, information nulling or parity puncturing can be conducted separately to achieve rate compatibility. However, neither technique works as well as the proposed hybrid technique when used over a wide range of rates. If only information nulling is employed, the high rate codes will contain a relatively large number of columns, leading to a large number of cycles and stopping sets. These code structures negatively affect the performance of message passing decoders (see [RSU01]). If we use parity puncturing alone to get a rate 0.8 code from a rate 0.2 code, we will have to puncture 75% of the bits (*i.e.*, 93.75% parity bits). Such heavy puncturing significantly increases the error floor because a large punctured set increases the chance of punctured stopping sets, and punctured stopping sets cannot be corrected ([DPT$^+$02] [TJVWa]). In addition, heavy puncturing reduces the effective block length of high rate codes. Short block length harms code performance. In the above example, the rate 0.8 code only transmits 25% of the bits that are transmitted for rate 0.2. In contrast, for the proposed scheme, the rate 0.8 code transmits 62.5% of the total bits.

5.4 Simulation Results

Simulation results for Additive White Gaussian Noise (AWGN) channels are shown in Fig. 5.3. The degree distribution profile of the mother code is described by Fig. 5.2. The mother code is generated by the ACE algorithm with the further constraint that columns be allocated per the degree allocation technique of the previous section. The parity matrix is also constructed to have a semi-lower triangular form as this prevents stopping set activation due to parity puncturing.

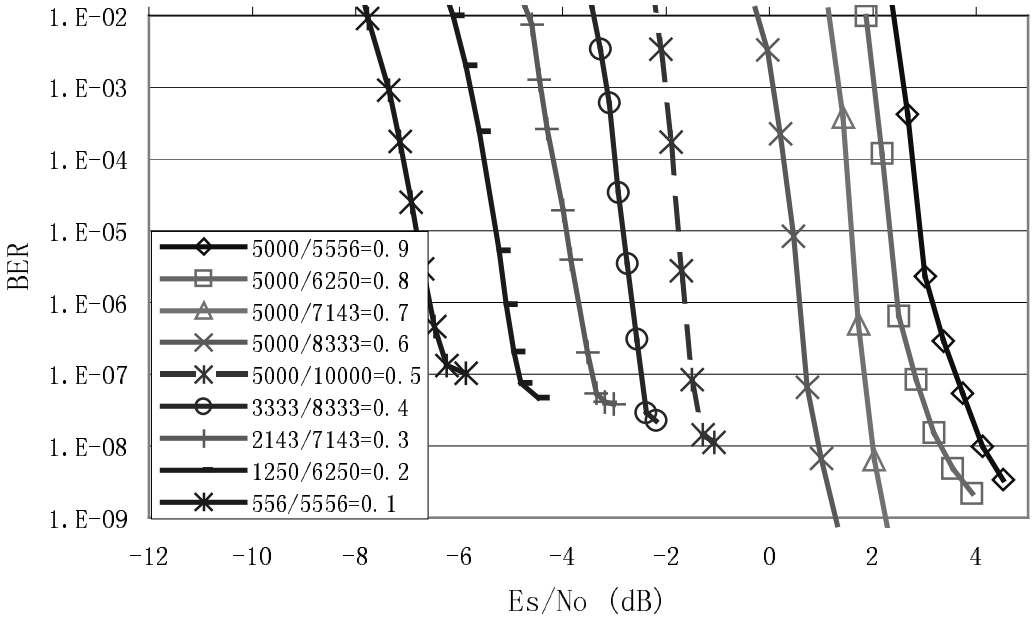


Figure 5.3: E_s/N_0 simulation results. AWGN channel.

It should be noted that when we applied the ACE algorithm to the multi-rate scheme, we enforced lower rate codes to have higher ACE parameters. We enforce $d_{ACE} = 9$ and set η_{ACE} to be 6, 5, 4 and 3 for $R = 0.1$, 0.2, 0.3 and 0.4 respectively. This step is crucial to further suppress the error floor of low rate code schemes.

The error floors of $R = 0.6$ and 0.7 are too low to be detected. Highest rates $R = 0.8$ and 0.9 do not show a distinct error floor. Instead, their curves transit gradually from a steep waterfall region to the error floor region. The mother code $R = 0.5$ has an error floor just above 10^{-8}. With the decrease of code rate, higher error floors are observed.

Fig. 5.4 plots Shannon capacity and the E_b/N_0 needed for the simulation code to reach BER $= 10^{-6}$. It shows an asymmetric degradation around $R_0 = 0.5$.

On the high rate side, $R = 0.6$ has a larger gap to capacity than other codes. This is because the $R = 0.6$ parity matrix has a larger portion of active triangular columns, which reduces the randomness of the code. In contrast, the highest rate code $R = 0.9$, has no active triangular columns. In addition, high rate codes enjoy smaller performance

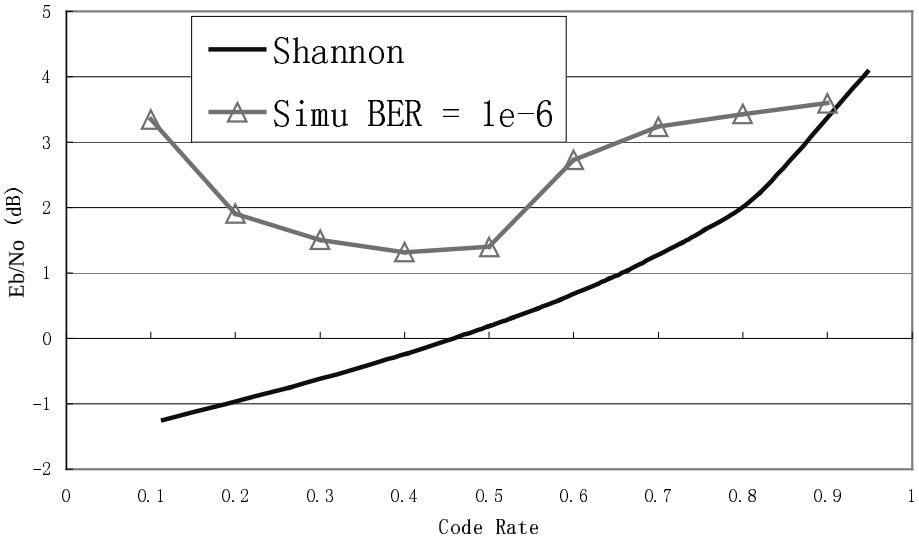


Figure 5.4: Gap to BPSK capacity bound.

gap because coded systematic bitstreams asymptotically approach the uncoded bitstream at $R \to 1$.

On the low rate side, the gap to capacity increases more rapidly as the code rate decreases. The reason is two-fold. First, low rate codes are shorter, *e.g.*, code $R = 0.1$ has only 5556 bits instead of 10000 bits. Second, when the normalized degree distribution (Fig. 5.2) was constructed, the distribution around $R = 0.1$ was changed slightly to satisfy (5.2). Therefore, the deviation of low rate distributions from the density evolution distribution is larger.

5.5 Summary

We combined information nulling and parity puncturing techniques for LDPC codes. The proposed scheme achieves close-to-capacity performance across a wide range of code rates.

Chapter 6

Compression of Correlated Sources Using LDPC Codes

In previous chapters, we have discussed LDPC codes used in channel coding scenarios. In this chapter, we will further apply LDPC codes to compressing memory correlated binary sources, where the correlation between sources is defined by a Hidden Markov Model (HMM). In order to achieve this goal, the standard approach in density evolution is modified to incorporate the HMM. The proposed scheme is then applied to the design of irregular LDPC codes that optimize the system performance.

6.1 Finite-State Markov Channels and Gilbert-Elliott Channels

A Finite-state Markov Channel (FSMC) can be modelled as a markov chain S_n which takes values in a finite space C of memoryless channels with finite input and output alphabets. The conditional input/output probability is $p(y_n|x_n, S_n)$, where x_n and y_n denote the channel input and output respectively. The channel transition probabilities are independent of the input. If the transmitter and receiver have perfect channel state information (CSI),

then the capacity of the FSMC is just the statistic average over all states of the corresponding channel capacity [MBD89]. On the other hand, with no CSI or the transition structure, capacity is reduced to that of the Arbitrarily Varying Channel [CK81]. An intermediate case was considered in [GV96], where the channel transition structure of the FSMC is known.

A correlated source pair can be defined on a binary FSMC as follows. The correlation between the sources is generated in the following way:

- Generate a symmetric i.i.d. sequence $\mathbf{U^1}$ ($P(u_k^1 = 0) = P(u_k^1 = 1) = 1/2$).
- Define the sequence $\mathbf{U^2}$ as $u_k^2 = u_k^1 \oplus e_k$, where $\oplus$ indicates modulus 2 addition and e_k is a binary random variable generated by an HMM $\lambda = \{A, B, \pi\}$. Let the set of states in this model be $S_j, 0 \leq j \leq S-1$. Defined on these states, there are three parameters in this model: $A = (a_{ij})$ is the matrix of transition probabilities among states (where $a_{ij} = P_t(S_j|S_i), 0 \leq i, j \leq S-1$ is the transition probability from state S_i to state S_j), $B = (b_{jv})$ is the list that gives the bit probability to associate with each state (where $b_{jv} = P_o(v|S_j), 0 \leq j \leq S-1,\ v \in \{0, 1\}$ is the probability of getting output v in state S_j), π is the initial distribution of each state.

As a special case of the FSMCs, a Gilbert-Elliott (G-E) channel has only two states: a 'good' state (S_0) and a 'bad' state (S_1). The input and output alphabets are binary and the channel at any specific state is a BSC. The BER at the 'good' state is lower than the BER at the 'bad' state.

For simplicity, we consider correlated sources defined on a G-E channel. There are four parameters in this model: b is the transition probability from S_0 to S_1, g is the transition probability from S_1 to S_0, p_G and p_B are the probabilities to generate a one in the 'good' and the 'bad' state respectively. This model is shown in Fig. 6.1 with parameters.

[MBD89] elaborated the relationship between channel capacities defined in three different ways. The capacity of the interleaved channel, under the assumption of no memory, is denoted by C^{NM}. This capacity is lower than C^{μ}, the capacity of the E-B channel, where

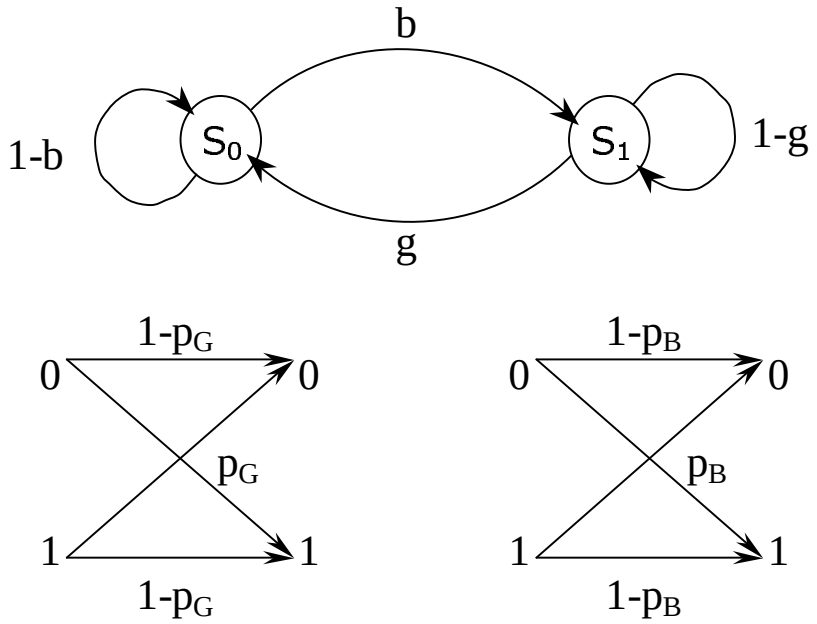


Figure 6.1: The E-B channel model.

$\mu = 1 - b - g$ is called the memory measure. C^{μ} in turn is lower than C^{SI}, which is the capacity assuming that the CSI is perfectly available to the transmitter and the receiver. For highly persistent channels ($\mu \to 1$) and highly oscillatory channels ($\mu \to -1$), C^{μ} is very close to C^{SI}.

6.2 Encoding and Non-Zero Syndrome Decoding

The well-known Slepian-Wolf result [SW73] states that when source encoding is performed separately for each source and the correlation between the sources is also assumed to be unknown at the encoder, the achievable compression region is given by $R_1 \geq H(\mathbf{U}^1|\mathbf{U}^2)$, $R_2 \geq H(\mathbf{U}^2|\mathbf{U}^1)$, $R = R_1 + R_2 \geq H(\mathbf{U}^1, \mathbf{U}^2)$, where $H(\mathbf{U}^i|\mathbf{U}^j)$ is the conditional entropy, $H(\mathbf{U}^1, \mathbf{U}^2)$ is the joint entropy, and R_i is the compression rate for sources i.

In this work we develop a density evolution analysis of the system proposed in [GFZb]. We consider (as in [LXG02]) the case of asymmetric compression, in which one of the sources is perfectly available to the decoder and the other source is compressed as much

as possible. Without loss of generality, we will assume that $R_2 = 1$ and source 1 is to be compressed to a rate R_1 as close to $H(\mathbf{U}^1|\mathbf{U}^2)$ as possible. Since from an analytical perspective the compression of $\mathbf{U}^1$ is equivalent to the compression of $\mathbf{U}^1 \oplus \mathbf{U}^2$ when $\mathbf{U}^2$ is known, we will focus on the decoding of the correlation pattern $\mathbf{U}^1 \oplus \mathbf{U}^2$. The encoding diagram is described in Fig. 6.2.

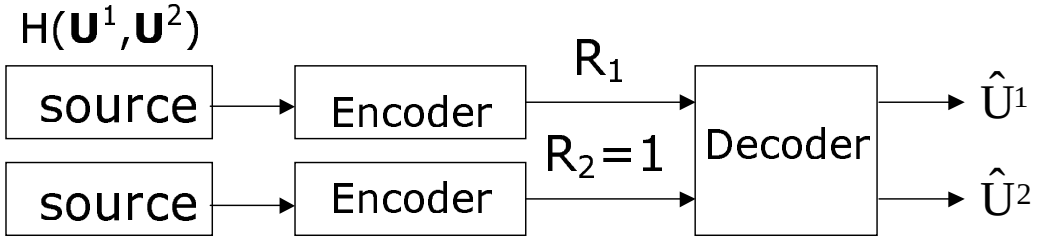


Figure 6.2: Encoding process of correlated sources.

The Slepian-Wolf limit states that 'neither encoder knows the other sequence being encoded'. This requirement can be achieved by the encoding and non-zero syndrome decoding technique described below:

Encoding and Non-Zero Syndrome Decoding

1. Encoder 2 compresses sequence $\mathbf{U^2}$ into bitstream $\mathbf{c_2}$.
2. Encoder 1 calculates the syndrome of sequence $\mathbf{U^1}$ as $\mathbf{s_1} = \mathbf{U^1 H^T}$, where $\mathbf{H}$ is the LDPC parity matrix.
3. Decoder decompresses bitstream $\mathbf{c_2}$ into sequence $\mathbf{U^2}$.
4. Decoder calculates the syndrome of sequence $\mathbf{U^2}$ as $\mathbf{s_2} = \mathbf{U^2 H^T}$.
5. Decoder decodes syndrome $\Delta\mathbf{s} = \mathbf{s_1} - \mathbf{s_2}$ by finding a sequence $\Delta\mathbf{U}$ that satisfies $\Delta\mathbf{s} = \Delta\mathbf{U}\mathbf{H^T}$.
6. Decoder finds $\mathbf{U^1} = \mathbf{U^2} + \Delta\mathbf{U}$.

It should be noted that step 5 is not error-free. By optimizing the parity matrix of the LDPC code, we can approach the Slepian-Wolf limit at very low error rate.

Fig. 6.3 shows the block diagram of the correlated source decoder.

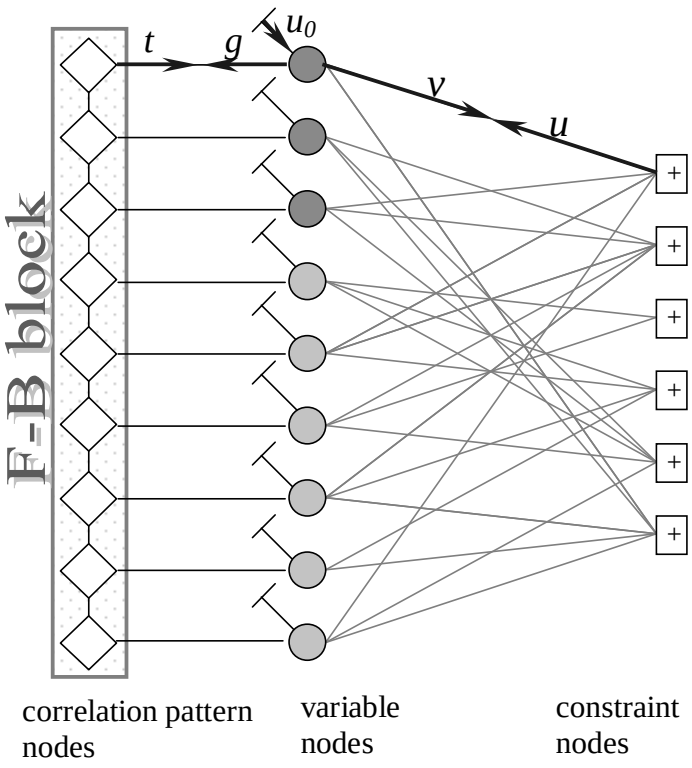


Figure 6.3: Message passing in the correlated source decoder.

In addition to n variable nodes and $n-k$ constraint nodes, the decoder has n correlation pattern nodes. These nodes comprise a block where the forward-backward (F-B) algorithm is used based on the transition trellis. Not including the F-B block, there are five types of messages passing in this decoding structure:

1. u messages (from constraint nodes to variable nodes)

2. v messages (from variable nodes to constraint nodes)

3. u_0 messages (*a priori* messages based on observation of channel corrupted signals)

4. g messages (from variable nodes to correlation pattern nodes)

5. t messages (from correlation pattern nodes to variable nodes)

The first three types of messages exist in the classic LDPC decoder. The other two types of messages (g and t) are related to the correlation of the source model. According to the principle of message passing, for an 'equality' function node, the output LLR should be the sum of all the input LLRs (see [For01]). A variable node is by nature an 'equality' function node, therefore,

$$g = \sum_{j=0}^{d_v} u_j. \tag{6.1}$$

The F-B block processes the input g messages based on the correlation model (b, g, p_G, p_B). The output t messages of the F-B block will be further combined with u_0 and u to generate the outgoing v messages of variable nodes:

$$v = t + \sum_{j=0}^{d_v-1} u_j. \tag{6.2}$$

The remaining problem is to model the relationship between the input and output of the F-B block, *i.e.*, to find the function f that defines $t = f(g)$. The next section will solve this problem.

6.3 LLR Evolution in the Forward-Backward Algorithm

In order to design a density evolution scheme for correlated sources in which the correlation is defined by HMMs [BCJR74], we need to understand how LLRs evolve in the F-B block. Given that the k^{th} correlation pattern bit $e^{(k)}$ is one with extrinsic probability $p^{(k)}$, we need to update this probability for the next iteration (denoted as $p'^{(k)}$). The basic unit in the F-B trellis is shown in Fig. 6.4,

Define the forward state probability to be a row vector $\alpha^{(k)} = [\alpha_0^{(k)}\alpha_1^{(k)}]$, where $\alpha_i^{(k)}$ is the probability that the k^{th} correlation pattern bit is in state S_i. Similarly we define the backward state probability to be a column vector $\beta^{(k)} = [\beta_0^{(k)}\beta_1^{(k)}]^T$. The boundary

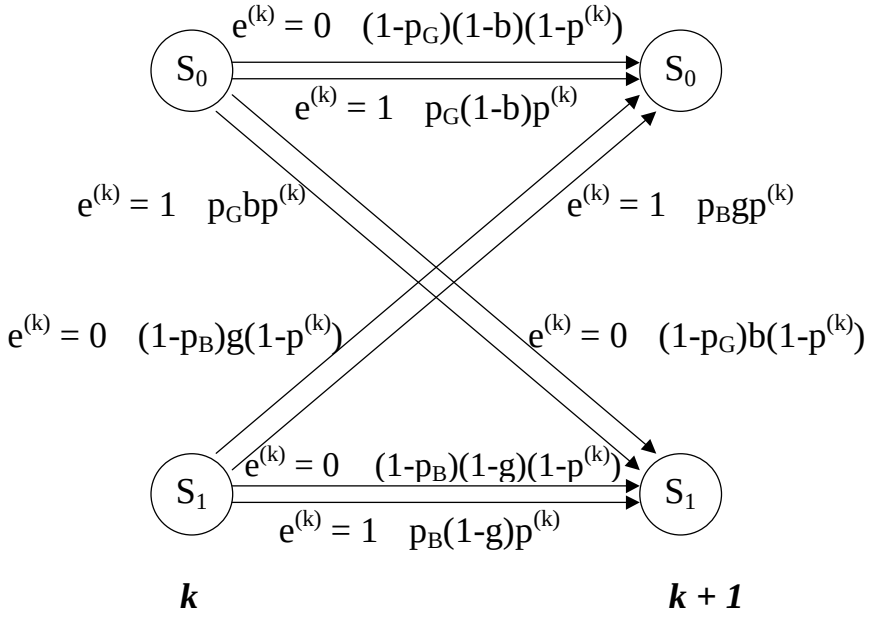


Figure 6.4: The basic unit in the F-B trellis

conditions can be set to be the stationary distribution: $\alpha^{(0)} = \beta^{(n)T} = \left[\begin{array}{cc} \frac{g}{g+b} & \frac{b}{g+b} \end{array} \right]$. One step in the F-B algorithm can be described as

$$\alpha^{(k+1)} = \alpha^{(k)} Q^{(k)}, \tag{6.3}$$

and

$$\beta^{(k+1)} = Q^{(k)} \beta^{(k)}, \tag{6.4}$$

where

$$Q^{(k)} = \left[\begin{array}{cc} Q_{00}^{(k)} & Q_{01}^{(k)} \\ Q_{10}^{(k)} & Q_{11}^{(k)} \end{array} \right], \tag{6.5}$$

$Q_{00}^{(k)} = (1 - p_G)(1 - b)\left(1 - p^{(k)}\right) + p_G (1 - b) p^{(k)},$
$Q_{01}^{(k)} = (1 - p_G) b \left(1 - p^{(k)}\right) + p_G b p^{(k)},$
$Q_{10}^{(k)} = (1 - p_B) g \left(1 - p^{(k)}\right) + p_B g p^{(k)},$
$Q_{11}^{(k)} = (1 - p_B)(1 - g)\left(1 - p^{(k)}\right) + p_B (1 - g) p^{(k)}.$

The F-B updating rule is given by

$$p_0'^{(k)} = \alpha^{(k)} \begin{bmatrix} (1-p_G)(1-b) & (1-p_G)\,b \\ (1-p_B)\,g & (1-p_B)(1-g) \end{bmatrix} \beta^{(k)}, \tag{6.6}$$

$$p_1'^{(k)} = \alpha^{(k)} \begin{bmatrix} p_G(1-b) & p_G b \\ p_B g & p_B(1-g) \end{bmatrix} \beta^{(k)}, \tag{6.7}$$

$$p'^{(k)} = \frac{p_0'^{(k)}}{p_0'^{(k)} + p_1'^{(k)}}. \tag{6.8}$$

By using likelihood ratios instead of probabilities, a simpler representation of the F-B algorithm is obtained as follows

$$\Gamma_0 = \begin{bmatrix} (1-p_G)(1-b) & (1-p_G)\,b \\ (1-p_B)\,g & (1-p_B)(1-g) \end{bmatrix}, \tag{6.9}$$

$$\Gamma_1 = \begin{bmatrix} p_G(1-b) & p_G b \\ p_B g & p_B(1-g) \end{bmatrix}, \tag{6.10}$$

$$l^{(k)} = \frac{1-p^{(k)}}{p^{(k)}}, \tag{6.11}$$

$$Q^{(k)} = \Gamma_0 l^{(k)} + \Gamma_1, \tag{6.12}$$

$$l'^{(k)} = \frac{p_0'^{(k)}}{p_1'^{(k)}} = \frac{\alpha^{(k)}\Gamma_0\beta^{(k)}}{\alpha^{(k)}\Gamma_1\beta^{(k)}} = \frac{\begin{bmatrix} g & b \end{bmatrix} \prod_{i=0}^{k-1} Q^{(i)}\Gamma_0 \prod_{j=k+1}^{n} Q^{(j)} \begin{bmatrix} g \\ b \end{bmatrix}}{\begin{bmatrix} g & b \end{bmatrix} \prod_{i=0}^{k-1} Q^{(i)}\Gamma_1 \prod_{j=k+1}^{n} Q^{(j)} \begin{bmatrix} g \\ b \end{bmatrix}} \tag{6.13}$$

$$\approx \lim_{k\to\infty} \frac{\begin{bmatrix} g & b \end{bmatrix} \prod_{i=0}^{k-1} Q^{(i)}\Gamma_0 \prod_{j=k+1}^{2k} Q^{(j)} \begin{bmatrix} g \\ b \end{bmatrix}}{\begin{bmatrix} g & b \end{bmatrix} \prod_{i=0}^{k-1} Q^{(i)}\Gamma_1 \prod_{j=k+1}^{2k} Q^{(j)} \begin{bmatrix} g \\ b \end{bmatrix}}, \tag{6.14}$$

where (6.14) is obtained by neglecting the "edge effect" at the beginning and the end of a code (note that an LDPC code usually contains more than 10^3 bits).

For generic symmetric channels (see [RSU01]), the distribution of a variable output LLR message is well approximated by a Gaussian distribution whose mean (m) and variance (var) are related as $var = 2m$. If the channel is memoryless, we can use an all-zero codeword to test the performance of all possible codewords. Thus the Gaussian distribution in the test condition always has a positive mean with BPSK mapping: $0 \to +1/1 \to -1$.

Our HMM model is characterized by bit-flipping. It follows that the initial LLR distribution is impulsive (see [RSU01]). However, after a few iterations, the LLR distribution tends to be like a Gaussian. A caveat is that the traditional Gaussian approximation cannot be directly applied when we use the non-zero-syndrome decoding for the HMM model: we have to consider the sign of a correlation pattern bit. If the bit is 0, its F-B input Gaussian has a positive mean m; if the bit is 1, its F-B input Gaussian has a negative mean $-m$. The variance and m still satisfy $var = 2m$. To provide a normalized LLR, we change the sign of the output LLR if its associated correlation pattern bit is 1. The resulting metric represents the log-likelihood reliability of the output of the F-B algorithm.

Tab. 6.1 lists the parameters of three test sources. Source 1 is highly oscillatory, while source 2 and 3 are highly persistent. The memory measure μ and the theoretical compression rate R_1^{μ} were described in [MBD89]. As mentioned before, we assume $R_2 = 1$. It should be noted that all these sources have 50% ones on average, therefore they are incompressible with memoryless encoders. In this case, all the u_0 messages in Fig. 6.3 are zero.

source	b	g	p_G	p_B	μ	R_1^μ	Δ_{est}	Δ_{simu}
1	0.99	0.935	0.05	0.925	-0.925	0.515	1.82	2.05
2	0.03	0.033	0.07	0.973	0.937	0.448	2.22	2.42
3	0.01	0.011	0.055	0.9895	0.979	0.279	3.05	3.28

Table 6.1: Statistics of the test sources.

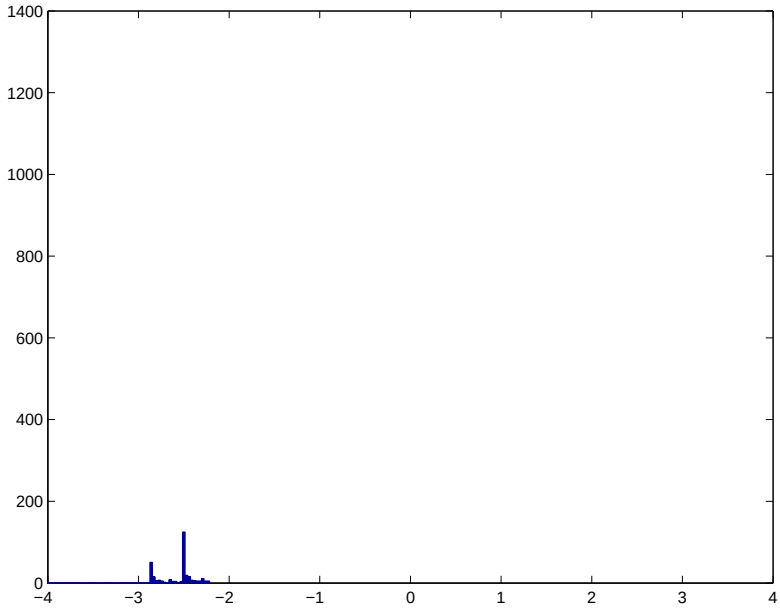
The LLR simulation results using 5000 bits for source 1 are shown in Fig. 6.5. Obviously, the output LLR distribution looks like two impulses with tails on their left sides. The mean of this distribution is positive, which represents an improvement in the reliability of correlation pattern nodes after an iteration. In contrast, without the F-B block, the *a priori* LLRs ($\bar{u}_0$) of source 1 is zero. We also see that for higher input LLR levels, the output LLR distribution has smaller tails and tends to have finer granularity.

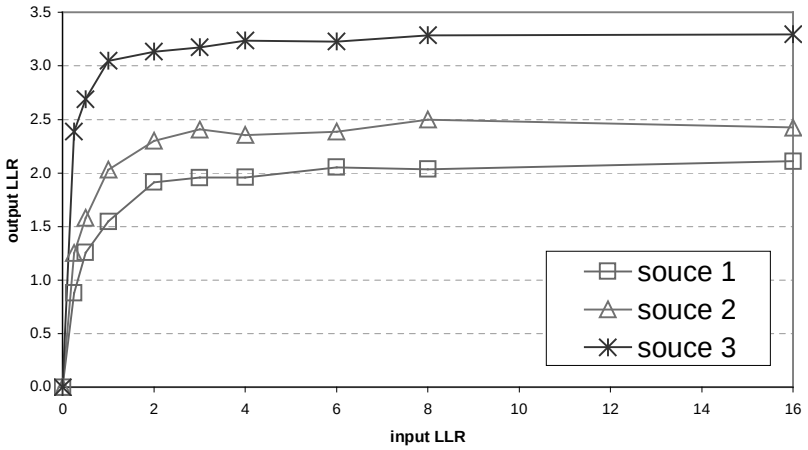
The F-B input-output characteristics of all the test sources are shown in Fig. 6.6(a). The output corresponding to zero input is $\bar{u}_0 = 0$. As the input level increases, the output LLR saturates to a value that we call Δ. In Tab. 6.1, Δ_{simu} represents the value of Δ obtained from simulation and Δ_{est} represents the estimate obtained with the method described in the Appendix.

As will be discussed in the next section, for sources with low *a priori* messages such as the test sources (where $\bar{u}_0 = 0$), the decoding process can hardly converge. In order to solve this problem, [GFZb] introduced periodic synchronization bits. Synchronization bits are directly transmitted to the decoder with complete reliability. For these bits (6.12) should be changed to

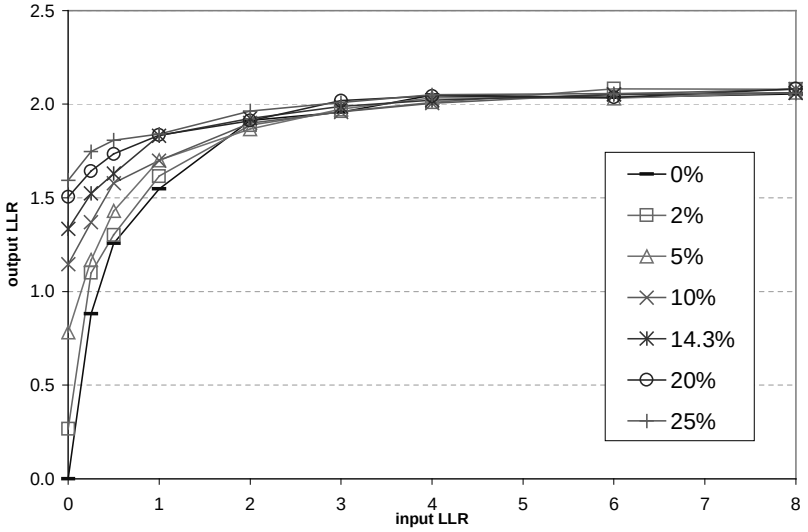
$$Q^{(k)} = \begin{cases} \Gamma_0, & \text{if this bit is 0,} \\ \Gamma_1, & \text{if this bit is 1.} \end{cases} \tag{6.15}$$

The decoder structure with synchronization bits can be illustrated in Fig. 6.7. The g messages passed from synchronization bits are either $+\infty$ or $-\infty$, because we have complete confidence in their values. Effectively, the whole F-B block is broken into smaller F-B blocks


1400
1200
1000
800
600
400
200
0
−4
−3
−2
−1
0
1
2
3
4



(a) All sources without synchronization bits



(b) Channel 1 with synchronization bits

Figure 6.6: Input-output characteristics of the F-B block generated by simulation.

marked by the synchronization bits. Each smaller F-B block has deterministic staring bit and ending bit.

Fig. 6.6(b) shows the Monte-Carlo simulation result for source 1 with different rate of synchronization bits. Note that the initial LLR output increases with the introduction of synchronization bits. Because synchronization bits introduce a transmission overhead, there is an optimal percentage of synchronization bits for given source parameters. This

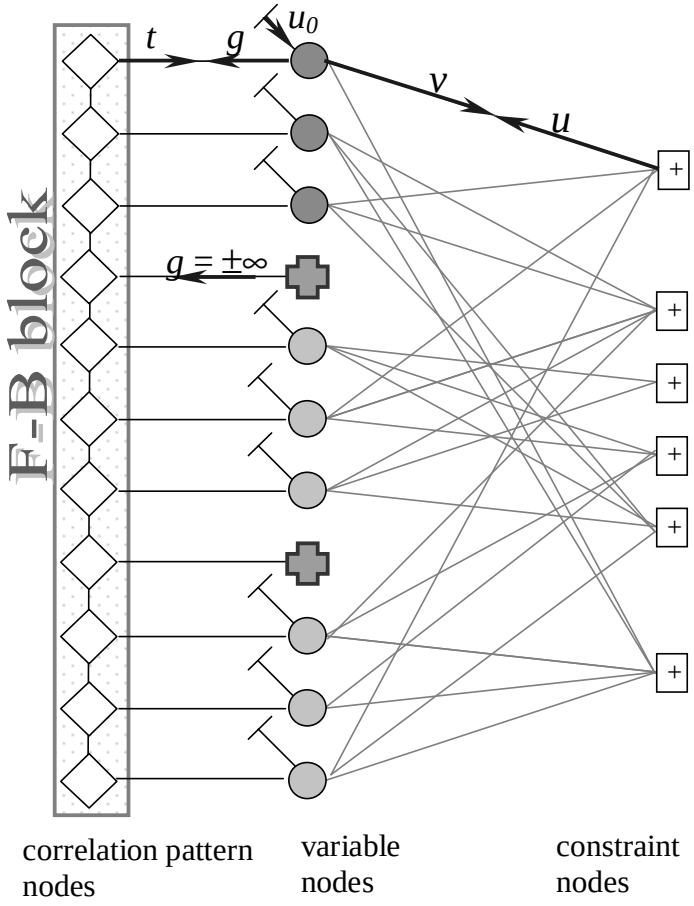


Figure 6.7: Message passing with two synchronization bits.

issue will be addressed shortly.

6.4 Density Evolution Optimization of Irregular LDPC Codes for Correlated Sources

Chung *et al.* [CRU01] designed a density evolution optimization algorithm based on linear programming and a Gaussian approximation. Divsalar *et al.* [DDP01] illustrated density evolution for various turbo-like codes by examining the iterative decoding tunnel between two SNR characteristic curves. The shape of this tunnel directly affects the convergence of an iterative decoder. With similar methodology, we propose an iterative decoding tunnel

for LDPC codes. This method is applicable to binary erasure channels (BECs) as well as binary-input additive white Gaussian noise channels (BI-AWGNCs). For simplicity, we only introduce the BI-AWGNC analysis here. First we reformulate Chung's density evolution equation [CRU01] as

$$\bar{u}_l = \sum_j \rho_j \Phi^{-1}\left\{\left[\sum_i \lambda_i \Phi\left(\bar{u}_0 + (i-1)\bar{u}_{l-1}\right)\right]^{j-1}\right\}, \tag{6.16}$$

$$\bar{u}_1 = 0, \text{ (initial condition)}, \tag{6.17}$$

where $\bar{u}_0 = \frac{2}{\sigma^2}$ is the mean of the *a priori* log-likelihood ratios (LLRs) and $\bar{u}_l$ is the mean of the LLRs generated by constraint nodes after the l^{th} iteration. The Φ function is defined to be

$$\Phi(x) = \begin{cases} \frac{1}{\sqrt{4\pi x}} \int_R \tanh\left(\frac{u}{2}\right) \exp\left(-\frac{(u-x)^2}{4x}\right) du, & \text{if } x > 0, \\ 0 & \text{if } x = 0. \end{cases} \tag{6.18}$$

To examine the LLR evolution in an LDPC code system, we separate (6.16) into two equations.

$$\bar{u}_l = \sum_j \rho_j \Phi^{-1}\left(\bar{T}_{l-1}^{j-1}\right), \tag{6.19}$$

$$\bar{T}_l = \sum_i \lambda_i \Phi\left(\bar{u}_0 + (i-1)\bar{u}_l\right), \tag{6.20}$$

where $\bar{T}_l = E\left(\tanh\frac{v_l}{2}\right)$, $\bar{v}_l$ is the mean of the LLRs generated by variable nodes after the l^{th} iteration. Equations (6.19) and (6.20) for an irregular rate 1/2 code (see [RSU01]) are plotted as the upper curve and the lower curve in Fig. 6.8.

An iterative decoding process begins at the origin and get through the tunnel bounded by the two curves in a manner similar to climbing a staircase. If the tunnel is open, the decoding process succeeds with infinite LLR achieved. If there are bottlenecks inside the tunnel, the LLR will block at the first (lowest) bottleneck.

The upper curve is fixed for given degree distributions and it always passes the origin.

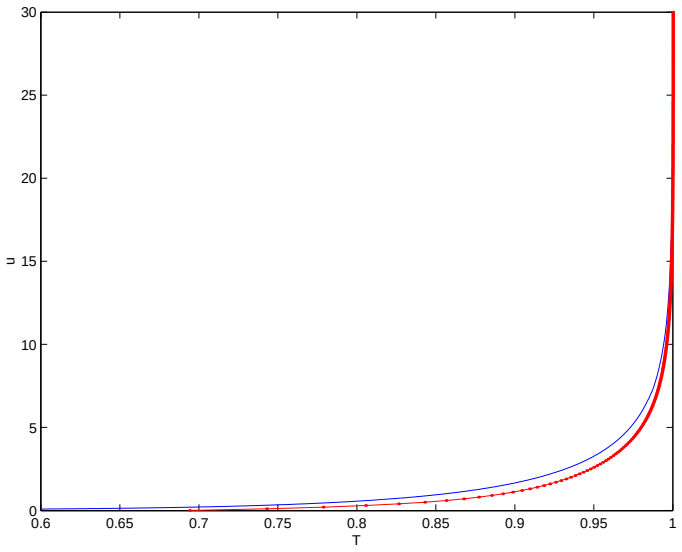


Figure 6.8: Density evolution curves for an irregular LDPC code. $d_v = 12$, $\sigma = 0.80$. The upper curve represents (6.19) and the lower curve represents (6.20).

The position of the lower curve depends on both degree distributions and channel parameters. This curve passes point ($\bar{T} = \Phi(\bar{u}_0)$, $\bar{u} = 0$). As the noise power (σ^2) increases, $\bar{u}_0$ decreases and the two curves become closer to form a narrower tunnel. The threshold noise power of a code is evaluated when the two curves touch. By imposing the constraint that the iterative decoding tunnel is open, we can design the following linear programming algorithm that optimizes the degree profile of irregular LDPC codes.

Traditional Optimization Algorithm:

For fixed ρ, maximize $\frac{1}{1-R} = \frac{\int \lambda}{\int \rho}$ such that

$\sum_{j=2}^{d_v} \lambda_j = 1$, $\lambda_j \geq 0$,

$\sum_i \lambda_i \Phi\left(\bar{u}_0 + (i-1)\bar{u}\right) > \bar{T}$ for many $\left(\bar{T}, \bar{u}\right)$ pairs that satisfy $\begin{cases} \bar{u} = \sum_j \rho_j \Phi^{-1}\left(\bar{T}^{j-1}\right) \\ \Phi\left(\bar{u}_0\right) < \bar{T} < 1 \end{cases}$,

$\lambda_2 < \frac{1}{\rho'(1)\exp\left(\frac{\bar{u}_0}{4}\right)}$.

The last inequality represents the stability constraint that enforces good code performance at high LLR (see [RSU01]). The above algorithm is similar to Chung's [CRU01] original optimization technique while providing a more intuitive depiction of the iterative

decoding tunnel.

In the standard density evolution process, variable nodes are connected to half edges representing *a priori* messages ($\bar{u}_0$ in (6.20)). The density evolution algorithm extends naturally to correlated sources. We only have to substitute $\bar{u}_0$ with the F-B output generated by the F-B input of the current iteration. For a degree-i variable this input is the sum of $\bar{u}_0$ and $i\bar{u}_l$, where $\bar{u}_l$ is the average LLR generated by constraint nodes in the l^{th} iteration. Thus the density evolution process for correlated sources can be described as

$$\bar{u}_l = \sum_j \rho_j \Phi^{-1}\left(\bar{T}_{l-1}^{j-1}\right) \ , \tag{6.21}$$

$$\bar{T}_l = \sum_i \lambda_i \Phi\left(f(\bar{u}_0 + i\bar{u}_l) + u_0 + (i-1)\bar{u}_l\right) \ , \tag{6.22}$$

where $f(\cdot)$ represents the input-output characteristics of the F-B block. The new density evolution algorithm that considers the F-B characterization is given by

Proposed Optimization Algorithm:

For fixed ρ, maximize $\frac{1}{1-R} = \frac{\int \lambda}{\int \rho}$

s.t.

$\sum_{j=2}^{d_v} \lambda_j = 1$,

$\lambda_j \geq 0$,

$\sum_i \lambda_i \Phi\left(f(\bar{u}_0 + i\bar{u}_l) + u_0 + (i-1)\bar{u}\right) > \bar{T}$ for many $\left(\bar{T}, \bar{u}\right)$ pairs that satisfy

$$\begin{cases} \bar{u} = \sum_j \rho_j \Phi^{-1}\left(\bar{T}^{j-1}\right) \\ \Phi\left(\bar{u}_0\right) < \bar{T} < 1 \end{cases},$$

$\lambda_2 < \frac{1}{\rho'(1)\exp\left(\frac{u_0}{4}\right)}$.

Recall that for sources with small *a priori* messages we use synchronization bits, because the lower curve in Fig. 6.8 starts from a point at or very close to the origin in this case. It should also be noted that the $f(\cdot)$ function in the proposed algorithm represents the F-B characteristics with corresponding percentage of synchronization bits.

6.5 Analytical and Simulation Results

For given source parameters, we can find the code that optimizes compression rate R_1. Let the fraction of synchronization bits be α, then the source-coding (compression) rate is $R_1 = (1 - R + \alpha)/(1 + \alpha)$. Fig. 6.9 shows the $R_1 \sim \alpha$ curves for source 1 (both the proposed and the traditional algorithm). Obviously, the proposed method achieves a better compression rate with less synchronization bits.

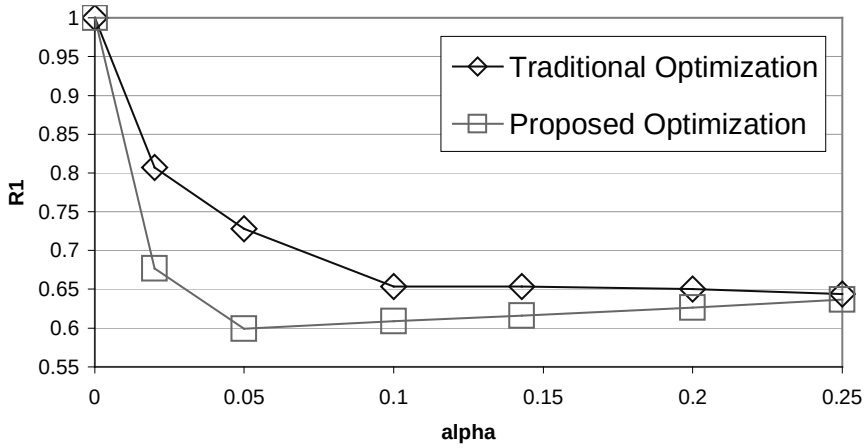


Figure 6.9: Theoretical $R_1 \sim \alpha$ curves for source 1.

Tab. 6.2 shows the optimum R_1 and α data for the test sources. For regular codes, we listed simulation results from [GFZb]. For irregular codes, we listed both theoretical thresholds and simulation results for the proposed technique. All the simulations are performed on codes with 16,400 bits. In general, the proposed algorithm generates irregular codes that outperform regular codes. However, for source (3), the irregular code performs worse than the best regular codes. The reason is that the Gaussian approximation used in the proposed algorithm is inaccurate for the first several iterations, which compromises the optimality of the proposed algorithm. Solution to this issue lies in an exact density evolution analysis without the Gaussian approximation.

We also noticed that the degree profile generated with the proposed algorithm is signif-

source	$(R_1, \alpha)^{reg}_{SIMU}$	$(R_1, \alpha)^{irreg}_{THEO}$	$(R_1, \alpha)^{irreg}_{SIMU}$
1	(0.666, 0.20)	(0.599, 0.05)	(0.634, 0.15)
2	(0.582, 0.20)	(0.544, 0.05)	(0.577, 0.13)
3	(0.423, 0.08)	(0.413, 0.02)	(0.434, 0.06)

Table 6.2: R_{comp} simulation result for regular codes and R_{comp} threshold for irregular codes.

icantly different from the one generated with the traditional algorithm. For source 1, the traditional algorithm yields

$$\lambda(x) = 0.21704x + 0.18592x^2 + 0.02477x^5 + 0.23672x^6 + 0.33554^{22}, \tag{6.23}$$

while the proposed algorithm yields

$$\lambda(x) = 0.19785x + 0.61618x^2 + 0.11814x^{10} + 0.06782^{30}. \tag{6.24}$$

Obviously the codes optimized for correlated sources have higher concentration on degree-3 variables in stead of degree-2 variables.

6.6 Appendix: Estimation of F-B Characteristics

If we consider only the two immediate neighbors of an correlation pattern bit, then (6.13) becomes

$$l'^{(k)} \approx \frac{\begin{bmatrix} g & b \end{bmatrix} Q^{(k-1)} \Gamma_0 Q^{(k+1)} \begin{bmatrix} g \\ b \end{bmatrix}}{\begin{bmatrix} g & b \end{bmatrix} Q^{(k-1)} \Gamma_1 Q^{(k+1)} \begin{bmatrix} g \\ b \end{bmatrix}}. \tag{6.25}$$

Source 1 is highly oscillatory ($\mu = 1 - b - g = 1 - 0.99 - 0.935 = -0.925$). The crossover probability of the 'good' state is low ($p_G = 0.05$) and that of the 'bad' state is very high ($p_B = 0.925$). Therefore, the two immediate neighbours of each correlation pattern bit tend to [illegible]

[illegible] $\frac{g}{g+b} \approx 0.486$ and $\frac{b}{g+b} \approx 0.514$ respectively. Thus for source 1, the mean of output LLRs is approximately $0.486 \times 2.70 + 0.514 \times 2.42 = .56$, as compared to 2.05.

[illegible] accurate estimate (Δ_{est} [illegible]) obtained [illegible] taking a weighted sum

6.7 Summary

In this chapter, we developed a density evolution analysis for compression of correlated sources using irregular LDPC codes. This analysis involves a depiction of the iterative decoding tunnel and a linear programming optimization method. Simulation results corroborate the good prediction capabilities of the proposed analysis.

Bibliography

[AEH] D. M. Arnold, E. Eleftheriou, and X. Y. Hu. Progressive edge-growth Tanner graphs. in *Proc. IEEE Global Telecommun. Conf.,* San Antonio, TX, Nov. 2001, 2:995–1001.

[AG] A. Aaron and B. Girod. Compression with side information using turbo codes. in *Proc. Data Compression Conf.,* Snowbird, Utah, April. 2002.

[Bar95] A. S. Barbulescu. Rate compatible turbo-codes. *IEE Electronic Lett.*, 31:535–536, Mar. 1995.

[BCJR74] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inform. Theory*, 20:284–287, Mar. 1974.

[BDMP98] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara. Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding. *IEEE Trans. Inform. Theory*, 44:909–926, May 1998.

[BGT] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo codes. in *Proc. IEEE Int. Conf. Commun.,* Geneva, Switzerland, May 1993.

[BM] J. Bajcsy and P. Mitran. Coding for Slepian-Wolf problem with turbo codes. in *Proc. IEEE Global Telecommun. Conf.,* San Antonio, TX, Nov. 2001.

[CCG79] J. B. Cain, G. C. Clark, and J. M. Geist. Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding. *IEEE Trans. Inform. Theory*, 25:97–100, Jan. 1979.

[CK81] I. Csiszá and J. Körner. Information theory: coding theorems for discrete memoryless channels. *New York: Academic Press*, 1981.

[CRU01] S. Chung, T. Richardson, and R. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Trans. Inform. Theory*, 47:657–670, Feb. 2001.

[DDP98] S. Dolinar, D. Divsalar, and F. Pollara. Code performance as a function of block size. *TMO Progress Report*, 42(133), May 1998.

[DDP01] D. Divsalar, S. Dolinar, and F. Pollara. Iterative turbo decoder analysis based on density evolution. *IEEE J. Select. Areas Commun.*, 19:891–907, May 2001.

[DM98] M. C. Davey and D. J. C. MacKay. Low-density parity check codes over GF(q). *IEEE Commun. Lett.*, 2:165–167, June 1998.

[DPT+02] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke. Finite length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Trans. Inform. Theory*, 48:1570–1579, June 2002.

[For01] G. D. Forney. Codes on graphs: normal realizations. *IEEE Trans. Inform. Theory*, 47:520–548, Feb. 2001.

[Fos01] M. Fossorier. Iterative reliability-based decoding of low-density parity-check codes. *IEEE J. Select. Areas Commun.*, 19:908–917, May 2001.

[FW] C. Fragouli and R. D. Wesel. Bit vs. symbol interleaving for parallel concatenated trellis coded modulation. in *Proc. IEEE Global Telecommun. Conf.,* San Antonio, TX, Nov. 2001, 2:931–935.

[Gal62] R. G. Gallager. Low-density parity-check codes. *IRE Trans. Inform. Theory*, IT-8:21–28, Jan. 1962.

[Gal63] R. G. Gallager. *Low-density parity-check codes.* Cambridge, MA: MIT Press, 1963.

[GF] J. Garcia-Frias. Joint source-channel decoding of correlated sources over noisy channels. in *Proc. Data Compression Conf.,* Snowbird, Utah, Mar. 2001, pages 283–292.

[GFZa] J. Garcia-Frias and Y. Zhao. Data compression of unknown single and correlated binary sources using punctured turbo codes. in *Proc. 39th Allerton Conf. Commun., Control and Comput.,* Oct. 2001.

[GFZb] J. Garcia-Frias and W. Zhong. LDPC codes for compression of multi-terminal sourcs with hidden Markov correlation. to appear in *IEEE Commun. Lett.*

[GV96] A. J. Goldsmith and P. P. Varaiya. Capacity, multual information, and coding for finite-state Markov channels. *IEEE Trans. Inform. Theory*, 42:868–886, May 1996.

[Hag88] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their applications. *IEEE Trans. Inform. Commun.*, 36:389–400, April 1988.

[HM] J. Ha and S. W. McLaughlin. Analysis and design of punctured LDPCCs over Gaussian channel with erasures. in *Proc. Int. Symposium Inform. Theory,* Lausanne, Switzerland, June 2002, page 30.

[JTM$^+$] C. Jones, T. Tian, A. Matache, R. D. Wesel, and J. D. Villasenor. Robustness of LDPC codes on periodic fading channels. in *Proc. IEEE Global Telecommun. Conf.,* Taipei, Taiwan, Nov. 2002.

[KFL01] F. R. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47:498–519, Feb. 2001.

[KLF01] Y. Kou, S. Lin, and M. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Trans. Inform. Theory*, 47:2711–2736, Nov. 2001.

[KS] H. Kim and G. L. Stüber. Rate compatible punctured turbo coding for W-CDMA. in *Proc. IEEE Int. Conf. Personal Wireless Commun.,* Hyderabad, India, Dec. 2000, pages 143–147.

[LMSS01] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. Inform. Theory*, 47:585–598, Feb. 2001.

[LXG02] A. D. Liveris, Z. Xiong, and C. N. Georghiades. Compression of binary sources with side information at the decoder using LDPC codes. *IEEE Commun. Lett.*, 6:440–442, Oct 2002.

[Mac99] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45:399–431, Mar. 1999.

[Man74] D. M. Mandelbaum. An adaptive-feedback coding scheme using incremental redundancy. *IEEE Trans. Inform. Theory*, 20:388–389, May 1974.

[Mar82] G. A. Margulis. Explicit construction of graphs without short cycles and low density codes. *Combinatorica*, 2(1):71–78, 1982.

[MB] Y. Mao and A. H. Banihashemi. A heuristic search for good low-density parity-check codes at short block lengths. in *Proc. IEEE Int. Conf. Commun.,* Helsinki, Finland, June 2001.

[MB01] Y. Mao and A. H. Banihashemi. Decoding low-density parity-check codes with probabilistic scheduling. *IEEE Commun. Lett.*, 5:414–416, Oct. 2001.

[MBD89] M. Mushkin and I. Bar-David. Capacity and coding for the Gilbert-Elliott channels. *IEEE Trans. Inform. Theory*, 35:1277–1290, Nov. 1989.

[MWD] D. J. C. MacKay, S. T. Wilson, and M. C. Davey. Comparison of constructions of irregular Gallager codes. in *Proc. 36th Allerton Conf. Commun., Control and Comput.,* Sept. 1998.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann, 1988.

[PR] S. S. Pradham and K. Ramchandran. Distributed source coding using syndromes (DISCUS): design and construction. in *Proc. Data Compression Conf.,* Snowbird, Utah, Mar. 1999, pages 158–167.

[RSU01] T. Richardson, M. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:638–656, Feb. 2001.

[RU01] T. Richardson and R. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:638–656, Feb. 2001.

[Sha59] C. E. Shannon. Probability of error for optimal codes in a Gaussian channel. *Bell Syst. Tech. J.*, 38:611–656, Oct. 1959.

[SVZ98] S. Shamai (Shitz), S. Verdu, and R. Zamir. Systematic lossy source/channel coding. *IEEE Trans. Inform. Theory*, 44:564–579, Mar. 1998.

[SW73] D. Slepian and J. K. Wolf. Noiseless coding of correlated information sources. *IEEE Trans. Inform. Theory*, 19:471–480, July 1973.

[Tan81] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, IT-27:533–547, Sept. 1981.

[TGFZa] T. Tian, J. Garcia-Frias, and W. Zhong. Compression of correlated sources using LDPC codes. in *Proc. Data Compression Conf.,* Snowbird, Utah, Feb. 2003.

[TGFZb] T. Tian, J. Garcia-Frias, and W. Zhong. Density evolution analysis of correlated sources compressed with LDPC codes. in *Proc. Int. Symposium Inform. Theory,* Yokohama, Japan, June 2003.

[TJVWa] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel. Selective avoidance of cycles in irregular LDPC code construction. *submitted to IEEE Trans. Commun.*

[TJVWb] T. Tian, C. Jones, John Villasenor, and Rick Wesel. Construction of irregular LDPC codes with low error floors. in *Proc. IEEE Int. Conf. Commun.,* Anchorage, Alaska, May 2003.

[Var97] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Inform. Theory*, 43:1757–1766, Nov. 1997.

[Wib96] N. Wiberg. Codes and decoding on general graphs. *Ph.D. dissertation*, Linkping Univertisy, Linkping, Sweden, 1996.

[WLS] R. D. Wesel, X. Liu, and W. Shi. Periodic symbol puncturing of trellis codes. in *Proc. 31st Asilomar Conf. Signals, Systems and Comput.,* Nov. 1997.

[Wyn74] A. D. Wyner. Recent results in the Shannon theory. *IEEE Trans. Inform. Theory*, 20:2–19, Jan. 1974.

[YSB] J. Yedidia, E. Sudderth, and J. Bouchaud. Projection algebra analysis of error-correcting codes. in *Proc. 39th Allerton Conf. Commun., Control and Comput.,* Oct. 2001, pages 662–671.