Low-Density Parity-Check Codes

Trellis and Turbo Coding: Chapter 9
Webversion IEEE/Wiley
www.turbocoding.net

Christian Schlegel
Department of Electrical Engineering
University of Alberta
Edmonton, Alberta, CANADA
Email: schlegel@ee.ualberta.ca
# Table of Contents

1. Table of Content ................................................................................. 1  
2. Introduction ......................................................................................... 2  
   1.1 LDPC Codes and Graphs ................................................................. 3  
   1.2 LDPC Decoding via Message Passing ............................................. 6  
2. Analysis Techniques ............................................................................ 8  
   2.1 Error Probability Evolution for Binary Erasure Channels .............. 9  
   2.2 Error Mechanism for LDPCs on BECs .......................................... 13  
   2.3 Binary Symmetric Channels and the Gallager Algorithms .......... 14  
   2.4 The AWGN Channel ................................................................... 17  
   2.5 LDPC Code Design via EXIT Charts ......................................... 24  
3. The Error Floor Problem ................................................................... 27  
   3.1 Exploiting Variable Node Degrees ................................................ 28  
   3.2 The ACE Construction ................................................................. 28  
4. LDPC Encoding ................................................................................... 31  
   4.1 Approximate Triangularization ....................................................... 31  
   4.2 Triangular LDPC Codes ................................................................. 33  
5. Repeat Accumulate Codes ................................................................. 36  
References ............................................................................................. 38
1 Introduction

Low-density parity-check (LDPC) codes can rightfully take their stand next to turbo codes as the most powerful error control codes known. They offer performances spectacularly close to theoretical limits when decoded using iterative soft-decision decoding algorithms based on factor graphs ([30], Chapter 8). Indeed, a rate \( R = 1/2 \) LDPC code with a blocklength of \( 10^7 \) bits has been shown to perform within 0.04 dB of the Shannon limit for the binary input additive white Gaussian noise channel at a bit error rate of \( 10^{-6} \) [7], and has a threshold only 0.0045 dB away from that limit. Both LDPC codes and turbo codes are capacity-approaching codes in that sense. They are both theoretically and practically capable of solving Shannon’s channel coding problem.

Low-Density Parity Check (LDPC) codes and a corresponding iterative decoding algorithm were first introduced by Gallager more than forty years ago [13, 14]. However, for the next several decades LDPC codes were largely forgotten, possibly because computers of the time could not simulate the performance of these codes with meaningful blocklengths at low error rates. Following the discovery of turbo codes ([30], Chapter 10), LDPC codes were rediscovered through the work of MacKay and Neal [25, 26] and have become a major research topic. LDPC codes significantly differ from the more conventional trellis and block codes. First, they are constructed in a random manner, and second, they have a decoding algorithm whose complexity is linear in the block length of the code, which allows the decoding of large codes. When combined with their spectacular performance, this makes LDPC codes a compelling class of codes, and their recent adoption in several standards, such as IEEE 802.16, IEEE 802.20, IEEE 802.3, and DVB-RS2, testifies to their bright future.

In this chapter, we discuss basic properties and results of low density parity check codes. We begin with the fundamental construction of regular LDPC codes and their representation as bipartite graphs. The graphical representation of LDPC codes then leads us to the notion of irregular LDPC codes and to the density evolution analysis technique which predicts the performance of these codes. LDPC codes and their decoders can be thought of as a prescription for selecting large random block codes, not unlike the codes suggested by Shannon’s coding theorem. MacKay [25] showed, by applying random coding arguments to the random sparse generator matrices of LDPC codes, that such code ensembles can approach the Shannon limit exponentially fast in the code length. And even at moderate lengths, these codes provide impressive performance results, some of which are illustrated in Figure 1.

![Figure 1: Performance results for regular LDPC codes of rate \( R = 1/2 \) for varying block lengths.](image-url)
1.1 LDPC Codes and Graphs

A linear block code \( C \) of rate \( R = \frac{k}{n} \) can be defined in terms of a \( (n-k) \times n \) parity check matrix \( H = [h_1, h_2, \ldots, h_n] \), where each \( h_j \) is a column vector of length \( n-k \). Each entry \( h_{ij} \) of \( H \) is an element of a finite field \( GF(p) \). We will only consider binary codes in this chapter, so each entry is either a ‘0’ or a ‘1’ and all operations are modulo 2. The code \( C \) is the set of all vectors \( x \) that lie in the (right) nullspace of \( H \), i.e., \( H x = 0 \). Given a parity check matrix \( H \), we can find a corresponding \( k \times n \) generator matrix \( G \) such that \( GH^T = 0 \). The generator matrix can be used as an encoder according to \( x^T = u^T G \).

In its simplest guise, an LDPC code is a linear block code with a parity check matrix that is “sparse”, i.e., it has a small number of nonzero entries. In [13, 14], Gallager proposed constructing LDPC codes by randomly placing 1’s and 0’s in a \( m \times n \) parity check matrix subject to the constraint that each row of \( H \) had the same number \( d_v \) of 1’s and each column of \( H \) had the same number \( d_c \) of 1’s. For example, the \( m = 15 \times n = 20 \) parity check matrix shown in Figure 2 has \( d_v = 4 \) and \( d_c = 3 \) and defines a LDPC code with length \( n = 20 \). Codes of this form are referred to as regular \( (d_v, d_c) \) LDPC codes of length \( n \). In a \( (d_v, d_c) \) LDPC code, each information bit is involved in \( d_v \) parity checks and each parity check bit involves \( d_c \) information bits. The fraction of 1’s in the parity check matrix of a regular LDPC code is \( \frac{md_c}{mn} = \frac{d_c}{n} \), which approaches zero as the block length gets large and leads to the name low density parity check codes.

\[
H_1 = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

Figure 2: Gallager-type low-density parity check matrix \( H_1 \).

As a first order of business, we determine the rate of the code defined by \( H_1 \). Since the parity check matrix is randomly constructed, there is no guarantee that the rows are linearly independent and the matrix has full rank. Indeed, the rank of \( H_1 \) is \( 13 < m = 15 \) and this parity check matrix actually defines a code with rate \( R = \frac{7}{20} \). In general, such randomly constructed parity check matrices will not be full rank and \( m \neq n - k \).
We could of course eliminate linearly dependent rows to find a \((n - k) \times n\) parity check matrix, but the new matrix would no longer be regular. For LDPC codes with large \(n\) it is convenient to retain the original parity check matrix even if it is not full rank and to refer to \(1 - \frac{m}{n} = 1 - \frac{d_c}{d_v}\) as the design rate of the code.

Having defined a blueprint for a \((d_v, d_c)\) regular LDPC code, we are now left to construct a particular instance of a code. To do this requires the choice of \(d_v, d_c, n\) and \(k\) which are constrained by the relationship that, for a regular code \(md_c = nd_v\), i.e., the number of non-zero entries of \(H\) must be the same whether calculated by rows or by columns. Furthermore, \(d_v\) must be less than \(d_c\) in order to have a rate \(R\) less than 1. Assuming that the block length \(n\) and the code rate are determined by the application, it remains to determine appropriate values for \(d_v\) and \(d_c\). In [14], Gallager showed that the minimum distance of a “typical” regular LDPC code increases linearly with \(n\), provided \(d_v \geq 3\). Therefore, regular LDPC codes are constructed with \(d_v\) on the order of 3 or 4, subject to the above constraints. For large block lengths, the random placement of 1’s in \(H\) such that each row has exactly \(d_c\) 1’s and each column has exactly \(d_v\) 1’s requires some effort and systematic methods for doing this have been developed [25, 26].

An important advance in the theory of LDPC codes is made possible by Tanner’s [31] method of using bipartite graphs to provide a graphical representation of the parity check matrix. (As a consequence, the bipartite graph of a LDPC code is sometimes referred to as a Tanner graph.) A bipartite graph is a graph in which the nodes may be partitioned into two subsets such that there are no edges connecting nodes within a subset. In the context of LDPC codes, the two subsets of nodes are referred to as variable nodes and check nodes. There is one variable node for each of the \(n\) bits in the code and there is one check node for each of the \(m\) rows of \(H\). An edge exists between the \(i^{th}\) variable node and the \(j^{th}\) check node if and only if \(h_{ij} = 1\). The bipartite graph corresponding to the parity check matrix \(H_1\) is shown in Figure 3. In a graph, the number of edges incident upon a node is called the degree of the node. Thus, the bipartite graph of a \((d_v, d_c)\) LDPC code contains \(n\) variable nodes of degree \(d_v\) and \(m\) check nodes of degree \(d_c\).

![Bipartite graph for the (3, 4) regular LDPC code with parity check matrix \(H_1\).](image)

It is clear that the parity check matrix can be deduced from the bipartite graph and thus the bipartite graph can be used to define the code \(C\). We can therefore start talking about codes as defined by a set of variable nodes, a set of check nodes, and set of edges. Note that the pair \((d_v, d_c)\), together with the code length \(n\), specifies an ensemble of codes, rather than any particular code. This ensemble is denoted by \(C^n(d_v, d_c)\). Once the degrees of the nodes are chosen, we are still free to choose which particular connections are made in the graph.

A socket refers to a point on a node to which an edge may be attached. For example, we say...
that a variable node has $d_v$ sockets, meaning $d_v$ edges may be attached to that node. There will be a total of $n d_v$ sockets on variable nodes and $m d_c$ sockets on parity check nodes. Clearly the number of variable node sockets must be equal to the number of check node sockets and a particular pattern of edge connections can be described as a permutation $\pi$ from variable node sockets to check node sockets.

We have the following code definition:

A regular LDPC code is completely defined by a permutation $\pi(i)$ of the natural numbers $1 \leq i \leq d_v n$. The index $i$ refers to the socket number at the variable nodes, and $\pi(i)$ to the socket number at the check nodes to which socket $i$ connects.

Selecting a random code from the ensemble $C^n(d_v, d_c)$ therefore amounts to randomly selecting a permutation on $n d_v$ elements. Many permutations will result in a graph which contains parallel edges, i.e. in which more than one edge join the same variable and parity check nodes. Note that, in the parity check matrix, an even number of parallel edges will cancel. If they are deleted from the graph, then the degrees of some nodes will be changed and the code will cease to be a regular LDPC code. If they are not deleted, their presence renders the iterative decoding algorithms ineffective. We must therefore make the restriction that permutations leading to parallel edges are disallowed.

It was observed by Luby et. al. [22] that allowing the degrees of the different nodes to differ can provide improved performance of the codes by up to 0.5dB for moderate-length codes on an additive white Gaussian noise channel at rate $R = 1/2$. An irregular LDPC code cannot be defined in terms of the degree parameters $d_v$ and $d_c$. We must instead use degree distributions to describe the variety of node degrees in the graph. A degree distribution $\gamma(x)$ is a polynomial in $x$

$$\gamma(x) = \sum_i \gamma_i x^{i-1},$$

such that $\gamma(1) = 1$. The coefficients $\gamma_i$ equal the fraction of edges in the graph which are connected to a node of degree $i$. The code length $n$ and two degree distributions -- $\lambda(x)$ and $\rho(x)$ for the variable and check nodes, respectively -- are sufficient to define an ensemble $C^n(\lambda, \rho)$ of irregular LDPC codes, and an individual irregular code is specified by the following definition:

An irregular LDPC code is completely defined by a permutation $\pi(i)$ of the natural numbers $1 \leq i \leq d_v N$ from variable to check node socket numbers and two degree distributions $\lambda(x)$ and $\rho(x)$ for the variable nodes and check nodes, respectively.

In the above definition we shall always assume that the nodes are ordered in descending degree values, which then completely specifies a given code.

The number of variable nodes in an irregular code is given by

$$n = N_e \sum_i \frac{\lambda_i}{i},$$

since $\lambda_i/i$ is the number of nodes of degree $i$, each such node combining $i$ edges, and $N_e$ is the number of edges in the codes. Likewise, the number of check nodes is given by

$$m = N_e \sum_i \frac{\rho_i}{i}.$$
The design rate of an irregular code is given by the dimension of the parity check matrix, or the size of the bipartite graph as

$$R = \frac{n - m}{n} = 1 - \frac{m}{n} = 1 - \frac{\sum_i \rho_i}{\sum_i \lambda_i}.$$  \tag{4}

Naturally, we want to rule out parallel edges also for irregular codes.

1.2 LDPC Decoding via Message Passing

One of the principle advantages of LDPC codes is that they can be decoded using an iterative decoding algorithm whose complexity grows (only) linearly with the block length of the code. Now that we have a graphical representation of LDPC codes, we can decode them using belief propagation decoding algorithms (see [30, Section 8.7]). Belief propagation is one instance of a broad class of message passing algorithms on graphs as discussed in [20, 12, 1, 34]. Decoding algorithms for the binary erasure channel and the binary symmetric channel are discussed below in Sections 2.1 and 2.3, respectively.

All message passing algorithms must respect the following rule:

**Rule 1 (Extrinsic Information Principle)**

A message sent from a node $i$ along an edge $e$ cannot depend on any message previously received on edge $e$.

This rule was introduced with Turbo codes [5] as the extrinsic information of a symbol.

Before stating the algorithm, it is necessary to formulate the decoding problem. A LDPC code is constructed in terms of its parity check matrix $H$, which is used for decoding. To encode the information sequence $u$, it is necessary to derive a generator matrix $G$ such that $GH^T = 0$. Finding a suitable $G$ is greatly simplified if we first convert $H$ into the equivalent systematic parity check matrix $H_S = [A|I_{n-k}]$. As is well-known, the systematic generator matrix is given by

$$G_S = [I_k|A^T].$$

The information sequence is now encoded as $x^T = u^T G_s$. It is worth noting that encoding by matrix multiplication has complexity $O(n^2)$, since $G_s$ is typically no longer sparse, and that therefore, in general, LDPC codes have linear decoding complexity, but apparently quadratic encoding complexity. Methods for reducing the encoding complexity are therefore of great interest. These are discussed in Section 4.

The codeword $x$ is transmitted over an additive white Gaussian noise (AWGN) channel using BPSK modulation resulting in the received sequence $r = (2x - 1) + n$ (see e.g. [30, Chapter 2]). The optimal decoder for the AWGN channel is the MAP decoder that computes the log-likelihood ratio

$$\lambda(x_r) = \log \left( \frac{P(x_n = 1|y)}{P(x_n = 0|y)} \right),$$

and makes a decision by comparing this LLR to the threshold zero. As shown in [30, Chapter 8], belief propagation on a graph, even with cycles, can closely approximate the MAP algorithm and we state the decoding algorithm for LDPC codes on the AWGN channel using these results.

Let $A = \{-1, +1\}$ denote the message alphabet, $r_i \in \mathcal{R}$ the received symbol at variable node $i$, and $\lambda_i \in \mathcal{R}$ the decision at variable node $i$. A message from variable node $i$ to check node $j$ is
represented by $\mu_{i \rightarrow j} \in \mathcal{R}$, and a message from check node $j$ to variable node $i$ is $\beta_{j \rightarrow i} \in \mathcal{R}$. Let $V_{j \setminus i}$ be the set of variable nodes which connect to check node $j$, excluding variable node $i$. Similarly, let $C_{i \setminus j}$ be the set of check nodes which connect to variable node $i$, excluding check node $j$. The decoding algorithm is then as follows:

**Algorithm for Decoding LDPC Codes on AWGN Channels:**

**Step 1:** Initialize $\lambda_i = 2 / \sigma^2 r_i$ for each variable node.

**Step 2:** Variable nodes send $\mu_{i \rightarrow j} = \lambda_i$ to each check node $j \in C_i$.

**Step 3:** Check nodes connected to variable node $i$ send

$$\beta_{j \rightarrow i} = 2 \tanh^{-1} \left( \prod_{l \in V_{i \setminus j}} \tanh \left( \frac{\lambda_l}{2} \right) \right).$$  \hspace{1cm} (5)

**Step 4:** Variable nodes connected to check nodes $j$ send

$$\mu_{i \rightarrow j} = \sum_{l \in C_{i \setminus j}} \beta_{l \rightarrow i}.$$  \hspace{1cm} (6)

**Step 5:** When a fixed number of iterations have been completed or the estimated codeword $\hat{x}$ satisfies the syndrome constraint $H \hat{x} = 0$, stop. Otherwise return to Step 3.

Figure 4 gives simulation results for a regular LDPC code, an irregular LDPC code and a turbo code ([30, Chapter 10]) of similar complexity on the AWGN channel [27]. This figure clearly demonstrates the power of irregular LDPC codes in terms of performance. The theoretical motivation and design methodology for irregular codes is discussed in the next section. The irregular node degree, however, can greatly increase the implementation complexity, especially if very high node degrees are used.

The check node’s rule (5) is fairly complex. But for quantized messages it is possible to map LLRs to messages in such a way that the check node rule can be implemented with only some combinational logics and an adder. Such decoders provide very good performance with only a few bits of precision. Finite-precision node processing and low-complexity implementations are discussed in Section ??.

If we want to keep the complexity even simpler, we can use the so-called min-sum approximation, in which we can replace the above processing rule with

$$\beta_{j \rightarrow i} \approx \min_{l \in V_{i \setminus j}} (|\lambda_l|) \prod_{l \in V_{i \setminus j}} \text{sign} (\lambda_l).$$  \hspace{1cm} (7)

Some performance loss in the order of 0.5dB will result from this approximation – see Section ??, but this loss may be justified by the overall savings in decoder complexity.

It is interesting to muse that the APP algorithm used in iterative turbo decoding can itself be viewed as an instance of belief propagation between nodes which represent individual trellis sections. The analogy between LDPC and turbo codes can further be strengthened using Forney’s normal graphs [12], which provide the interesting comparison illustrated in Figure 5, comparing the normal graphs for LDPC codes and standard Turbo codes. Both codes (and most others) can be represented as graph structures joined by a randomly selected permutation $\Pi$. This graphical representation also exposes the Turbo decoding algorithm as an instance of message-passing decoding on code graphs.
2 Analysis Techniques

We now know how to define ensembles of LDPC codes and how to decode them via message passing, but how do we know which parameters to choose and which codes to pick for good performance? The authors of [27, 28] use an approach they term “density evolution” to compare the qualities of different ensembles of regular and irregular LDPC codes. Density evolution has its origin in the error probability evolution formulas of Gallager [13, 14] discussed in Section 2.3, and the modern extension to soft-output channels are, in essence, a sophisticated generalization of these early methods.

The main statement that density evolution can make about LDPC codes is the following: A LDPC code with a given degree distribution pair \((\lambda, \rho)\), operated on a channel with noise standard deviation \(\sigma\) has an associated threshold \(\sigma^*\). The threshold is analogous to the Shannon capacity in that the error-probability for a randomly chosen \((\lambda(x), \rho(x))\)-code used on this channel can be made arbitrarily small for a growing block size of the code if and only if \(\sigma < \sigma^*\). One degree-distribution pair is said to be better than another if its threshold is closer to the channel’s capacity limit, i.e., if it can tolerate a higher noise standard deviation \(\sigma^*\).

Threshold results for several ensembles on various channels are reported in [28]. The qualities of different ensembles of regular and irregular LDPC codes are studied and optimized via density evolution in [8, 27], which use linear numerical optimization techniques to find optimal degree distributions.

One design methodology in designing good codes is to choose the ensemble with the best threshold, from which we select a code with the largest length which can be accommodated in a given implementation. As is typical in random coding arguments, it turns out that almost all codes in the ensemble perform equally well. Code design may therefore consist of randomly sampling a few codes from the
ensemble and selecting the best among those. The actual code selection is strongly determined by the code’s performance in the error floor region, where density analysis is not applicable (see Section 3).

### 2.1 (Error) Probability Evolution for Binary Erasure Channels

We first illustrate the method of density evolution using the Binary Erasure Channel (BEC), shown in Figure 6. For regular code ensembles, it is possible to derive the exact solution for thresholds when a simple discrete message-passing algorithm is used for decoding [4]. The channel parameter for the BEC is the erasure probability $\varepsilon$.

Let $\mathcal{A} = \{-1, 0, +1\}$ denote the message alphabet, $r_i \in \mathcal{A}$ the received symbol at variable node $i$, and $d_i \in \mathcal{A}$ the decision at variable node $i$. A message from variable node $i$ to check node $j$ is represented by $\mu_{i \rightarrow j} \in \mathcal{A}$, and a message from check node $j$ to variable node $i$ is $\beta_{j \rightarrow i} \in \mathcal{A}$. Let $C_{j \setminus i}$
be the set of variable nodes which connect to check node \( j \), excluding variable node \( i \). Similarly, let \( V_{i,j} \) be the set of check nodes which connect to variable node \( i \), excluding check node \( j \). The decoding algorithm for the BEC proceeds as follows [23]:

### Algorithm for Decoding LDPC Codes on BEC Channels:

**Step 1:** Initialize \( d_i = r_i \) for each variable node. If \( r_i = 0 \) then the received symbol \( i \) has been erased and variable \( i \) is said to be unknown.

**Step 2:** Variable nodes send \( \mu_{i \rightarrow j} = d_i \) to each check node \( j \in V_i \).

**Step 3:** Check nodes connected to variable node \( i \) send \( \beta_{j \rightarrow i} = \prod_{l \in C_j \setminus i} \mu_{l \rightarrow j} \) to \( i \). That is, if all incoming messages are different from zero, the check node sends back to \( i \) the value that makes the check consistent, otherwise it sends back a zero for “unknown”.

**Step 4:** If the variable \( i \) is unknown, and at least one \( \beta_{j \rightarrow i} \neq 0 \), set \( d_i = \beta_{j \rightarrow i} \) and declare variable \( i \) to be known. (Note that known variables will never have to be changed anymore for the simple BEC).

**Step 5:** When all variables are known, stop. Otherwise go back to Step 2.

Under this algorithm, a check node can only output a non-zero message on an edge if all incoming messages on other edges are non-zero. If there are not enough known variables when decoding begins, then the algorithm will fail.

Density evolution analysis for this channel and algorithm amounts to studying the propagation of erasure probabilities. A message is called “incorrect” if it is an erasure and has a numerical value of zero. We are now interested in determining the density of erasures among variable nodes as the number of iterations \( l \to \infty \). First, we simplify the message alphabet to \( A' = \{0, 1\} \), where a zero denotes an erasure and a one means the variable is known. As given by the decoding algorithm, the output of a check node is known if all of its incoming messages are known, and the output of a variable node is known if one or more of its incoming (check-node) messages are known.

Let \( p_u^{(l)} \) be the probability of erasures at the variable nodes at iteration \( l \), and let \( p_v^{(l)} \) be the probability, or “density” of erasures at the check nodes at iteration \( l \). Then for a regular code, in which the node degrees are all equal, the following recursive relation can be established:

\[
  p_u^{(l-1)} = 1 - \left[ 1 - p_v^{(l-1)} \right]^{d_v-1}
\]

\[
  p_v^{(l)} = p_0 \left[ p_u^{(l-1)} \right]^{d_u-1},
\]

where \( p_0 = \varepsilon \) is the initial density of erasures. Equation (8) contains an important assumption, which is that the probabilities of node \( u \) or \( v \), do not depend on themselves, that is, there are no cycles in the local subgraph which could introduce such dependencies.
This condition is best illustrated by Figure 7. In this local subtree, the nodes at level \( l \) do not appear in any of the lower levels. If this is the case, all probabilities which are migrated upward from below can be assumed to be independent of those at higher levels. This is, at least approximately, the case for large LDPC codes whose loops in the code’s graph are large. Hence the necessity to design LDPC codes with large loops, since otherwise the dependencies introduced in the message passing will, in general, degrade performance.

Combining (8), we obtain the equation of a one-dimensional iterated map for the probability that any variable node remains uncorrected as

\[
p_{v}^{(l)} = f(p_{v}^{(l-1)}) = p_{0} \left( 1 - \left[ 1 - p_{v}^{(l-1)} \right]^{d_{c}-1} \right)^{d_{v}-1}.
\]  

(9)

Figure 8 shows the evolution of the erasure probability for the ensemble of (3,6) LDPC codes for a number of input erasure values \( p_{0} = \varepsilon \). The staircase trajectory moving left shows the values \( p_{v}^{(l)} \) for \( \varepsilon = 0.4 \) according to equation (8). The solid curves express the function \( p_{v}^{(l)} = f(p_{v}^{(l-1)}) \), computed by combining the formulas in (8), and illustrate the evolution of \( p_{v}^{(l)} \) over iterations. Evidently, if \( \varepsilon \) is larger than a certain value, termed the decoding threshold, there exists a non-zero fix-point \( f(\varepsilon, x) = x \), and the erasure probabilities no longer converge to zero.

For an irregular code with degree-distributions \( \lambda(x) \) and \( \rho(x) \), the degree of a particular node is a random variable. When this is taken into account, the relation (8) becomes [22]

\[
p_{u}^{(l-1)} = 1 - \sum_{i=1}^{d_{u}} \rho_{i} \left[ 1 - p_{v}^{(l-1)} \right]^{i-1} = 1 - \rho \left( 1 - p_{v}^{(l-1)} \right)
\]

\[
p_{v}^{(l)} = p_{0} \sum_{j=1}^{d_{v}} \lambda_{j} \left[ p_{u}^{(l-1)} \right]^{j-1} = p_{0} \lambda \left( p_{u}^{(l-1)} \right).
\]

(10)

It can be derived by the following simple argument: the probability that a check node’s output is
non-zero is given by

$$P\left(\beta^{(l-1)} \neq 0\right) = \sum_{i=1}^{d_v} P\left(\text{all } i \text{ inputs known } | \#\text{inputs } = i\right) P\left(\#\text{inputs } = i\right)$$

$$= \sum_{i=1}^{d_v} P\left(\beta^{(l-1)} \neq 0\right) i P\left(\#\text{inputs } = i\right), \quad (11)$$

where we invoked the assumption that all messages are independent and identically distributed. Because $p_u^{(l)}$ and $p_u^{(l-1)}$ represent the probability that messages are *unknown*, the first line of (10) is an immediate consequence.

The same reasoning is used to obtain the second line of (10):

$$P\left(\mu^{(l-1)} = 0\right) = \sum_{j=1}^{d_c} P\left(j \text{ inputs unknown } | \text{node deg } = j\right) P\left(\text{node deg } = j\right)$$

$$= \sum_{j=1}^{d_c} P\left(\beta^{(l-1)} = 0\right)^j P\left(\text{node deg } = j\right). \quad (12)$$

Finally, we assume that the probability that a check node has degree $i$ is equal to $\rho_i$. Similarly, the probability for a variable node to have degree $j$ is equal to $\lambda_j$. This gives rise to the polynomial form used in the right-hand side of (10).

We proceed to determine the threshold for a code ensemble with parameters $(\lambda, \rho)$ on the BEC, by determining the conditions under which the density of erasures converges to zero as $l \to \infty$. Based on (10) and appealing to Figure 8, this threshold happens where the probability evolution function touches the straight line at a non-zero point, i.e., where $f(\varepsilon, x) = x$. This threshold has to be expressed
as a supremum, and for irregular LDPC codes is given by \[4\]

\[\varepsilon^* = \sup \{ \varepsilon : f(\varepsilon, x) < x, \forall x \leq \varepsilon \}\]

where \(f(\varepsilon, x) = \varepsilon \lambda (1 - \rho (1 - x))\)

Error-free decoding is possible if and only if

\[x = \varepsilon \lambda [1 - \rho (1 - x)]\]

has no positive solutions for \(x \leq \varepsilon\).

Since any fixed point solution \(x\) corresponds to a unique value of \(\varepsilon\):

\[\varepsilon(x) = \frac{x}{\lambda [1 - \rho (1 - x)]}\]

which allows us to rewrite the threshold as

\[\varepsilon^* = \min \{ \varepsilon(x) : \varepsilon(x) \geq x \}\]

For a regular ensemble \((d_v, d_c)\), this minimum can be solved analytically. Substitute \(x = 1 - y\) and consider the derivative of \(\varepsilon(1 - y)\), given by

\[
\frac{d}{dy} \left\{ \frac{1 - y}{[1 - y^{d_c-1}]^{d_v-1}} \right\} = \frac{[(d_c-1)(d_v-1)-1]y^{d_c-1}-(d_v-1)(d_c-1)y^{d_c-2}+1=0,}{(17)}
\]

which we set to zero in order to find the maximum.

By Descarte’s Rule of Signs, the number of positive real roots for a polynomial cannot exceed the number of sign changes in its coefficients. The polynomial in (17) therefore has no more than two roots. One of those roots is \(y = 1\), the original fixpoint at \(x = 0\). Dividing (17) by \((y - 1)\) yields

\[[(d_v-1)(d_c-1)-1]y^{d_c-2} - \sum_{i=0}^{d_c-3} y^i = 0\]

If \(s\) is the positive real root of (18), then the threshold is

\[\varepsilon^* = \frac{1 - s}{(1 - s^{d_c-1})^{d_v-1}}\]

As an example, the threshold for the \((3, 6)\) regular ensemble is \(\varepsilon^* = 0.4294\), (compare Figure 8). This code has nominal rate \(R = 0.5\). The capacity limit for the BEC at this rate is \(\varepsilon_C = 1 - R = 0.5\).

### 2.2 Error Mechanism of LDPCs on BECs

Large LDPC codes are very effective at correcting erasures. Additionally, the error mechanism of LDPCs on BECs is well understood. There are specific erasure patterns that cannot be corrected. These patterns are related to what are known as *stopping sets*, [10] defined as follows:

A stopping set \(S\) is a set of variable nodes, all of whose neighboring check nodes are connected to \(S\) at least twice.
Figure 9 illustrates a stopping set in our example LDPC code from Figure 3.

Note that the union of two stopping sets is again a stopping set, which is straightforward to see. Therefore, any arbitrary set \( A \) contains a unique maximal stopping set, since if more than one exist in \( A \), their union, which is larger, also exists in \( A \).

If a stopping set \( S \) is erased, the decoding algorithm from Section 2.1 will fail, since in step 3 of the algorithm, no parity check node which is a neighbor of \( S \) can ever send out a corrected message, since all neighbors of \( S \) see at least two erasures from \( S \). The nodes in \( S \), in turn, receive messages only from those neighbors, and hence keep receiving erasures irrespective of the number of iterations performed. From this observation then, we can deduce the following lemma:

**Lemma 2** Erasure decoding will terminate at the unique maximal stopping set contained in the erased set of variables.

### 2.3 Binary Symmetric Channels and the Gallager Algorithms

The BEC is admittedly a somewhat artificial channel model, mostly popular with information theorists to find closed-form solutions which are harder to find for other channel models. A more realistic channel model is that of the binary symmetric channel (BSC) shown in Figure 10, which has two input \( \{0, 1\} \) and two outputs \( \{0, 1\} \). A transmission is successful if input equals output, which happens with probability \( 1 - \varepsilon \). Conversely, an error is the inversion of a transmitted bit and happens with probability \( \varepsilon \). This BSC has a capacity of \( 1 - h(\varepsilon) \), where \( h(\varepsilon) \) is the binary entropy function [9].

The following analysis was presented by Gallager [13, 14], but disappeared in obscurity, like LDPC codes themselves, until their recent rediscovery. Gallager’s basic algorithm, called Gallager A, operates as follows:
Gallager’s LDPC Decoding Algorithm A for BSC Channels:

Step 1: Initialize $d_i = r_i$ for each variable node.

Step 2: Variable nodes send $\mu_{i\rightarrow j} = d_i$ to each check node $j \in V_i$.

Step 3: Check nodes connected to variable node $i$ send $\beta_{j\rightarrow i} = \prod_{l \in C \setminus i} \mu_{l\rightarrow j}$ to $i$. That is, the check node sends back to $i$ the value that would make the parity check consistent.

Step 4: At the variable node $i$ if $\lceil d_v/2 \rceil$ or more of the incoming parity checks $\beta_{j\rightarrow i}$ disagree with $d_i$, change the value of variable node $i$ to its opposite value, i.e., $d_i = d_i \oplus 1$.

Step 5: Stop when no more variables are changing, or after a fixed number of iterations have been executed. Otherwise go back to Step 2.

After specifying the algorithm, we can apply Gallager’s probability evolution analysis. Assume that the probability of error at the variable nodes is $p^{(0)}_v$, where $p^{(0)}_v = \varepsilon$. A check node will signal a correct check back if and only if an even number of errors occur in its checked symbols. This happens with probability

\[
1 - p^{(t)}_v = \frac{1 + (1 - 2p^{(t)}_v)^{d_c-1}}{2}.
\]  

(20)

In turn, at the variable nodes, an error will be corrected only if $b = \lceil d_v/2 \rceil$ or more parity checks are unsatisfied, which has probability

\[
p^{(t)}_v \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left( \frac{1 + (1 - 2p^{(t)}_v)^{d_c-1}}{2} \right)^t \left( \frac{1 - (1 - 2p^{(t)}_v)^{d_c-1}}{2} \right)^{d_v-1-t},
\]

(21)

where the two power terms simply express the probability of $t$ correct parity checks, and $d_v - 1 - t$ incorrect checks, respectively.

---

\textsuperscript{1}Equation (20) can be derived by considering the binomial expansions $(1 + p)^{d_c-1}$ and $(1 - p)^{d_c-1}$ and adding them.
The new probability of error in a variable node is now given by

\[
p_v^{(l+1)} = p_v^{(l)} - p_v^{(l)} \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left( \frac{1+\left(1-2p_v^{(l)}\right)^{d_v-1}}{2} \right)^t \left( \frac{1-\left(1-2p_v^{(l)}\right)^{d_v-1}}{2} \right)^{d_v-1-t}
+ (1-p_v^{(l)}) \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left( \frac{1-\left(1-2p_v^{(l)}\right)^{d_v-1}}{2} \right)^t \left( \frac{1+\left(1-2p_v^{(l)}\right)^{d_v-1}}{2} \right)^{d_v-1-t},
\]

where the first term is the probability that an erroneous variable node setting is corrected, and the second term is the probability that a correct setting is corrupted by faulty parity check information.

For a simple (3,6) LDPC code, the error probability evolution equation (22) becomes very simple. Its shape is illustrated in Figure 11, which is similar to the erasure probability evolution for the BEC channel.

![Figure 11: Graphical illustration of the evolution of the variable node error probability \(p_v^{(l)}\).](image)

From Figure 11, or numerical computations, it is evident that the threshold value for a large (3,6) LDPC code on a BSC is \(\varepsilon^* = 0.04\), while the channel capacity threshold for rate \(R = 0.5\) is \(\varepsilon = 0.11\), from the BSC capacity formula, given by \(C_{\text{BSC}} = 1 - h(R)\). This illustrates that it is quite a bit harder to approach capacity on this more severe channel.

Gallager [13, 14] has fine-tuned his algorithms by allowing a maximization over the “vote” parameter \(b\) in (22). This improves the threshold for the rate \(R = 0.5\) LDPC codes to \(\varepsilon^* = 0.052\) for (4,8) LDPC codes.

Irregular LDPC codes have become popular because they can improve on the performance of regular codes. In fact, adding irregularity to the node degrees has been the major constructive contribution to these codes since their inception in the early 60s. In an irregular code, the “vote” parameter depends on the degree \(j\) of the variable node, i.e., if \(b_j\) checks nodes disagree with the variable node, it is changed to the new value in Step 4 in the decoding algorithm. A check node will now signal a
ANALYSIS TECHNIQUES

17

correct check back with probability

\[ 1 - p_v^{(l)} = \frac{1 + \rho \left( 1 - 2p_v^{(l)} \right)}{2}, \quad (23) \]

which is a straightforward extension of (20) treating the node degrees as random according to \( \rho(x) \).

Analogously, the probability that an erroneous variable is corrected is extended to

\[ p_v^{(l)} \sum_{j=1}^{d_v} \lambda_j \sum_{t=b_j}^{j-1} \binom{j-1}{t} \left( \frac{1 + \rho \left( 1 - 2p_v^{(l)} \right)}{2} \right)^t \left( \frac{1 - \rho \left( 1 - 2p_v^{(l)} \right)}{2} \right)^{j-1-t} g(t,j), \quad (24) \]

and the new iterated variable node error probability is

\[ p_v^{(l+1)} = p_v^{(l)} - \sum_{j=1}^{d_v} \lambda_j \left[ \sum_{t=b_j}^{j-1} \binom{j-1}{t} \left( p_v^{(l)} g(t,j) + (1 - p_v^{(l)}) g(j,t) \right) \right] \quad (25) \]

The best “vote” parameter can be found to satisfy [22]

\[ \frac{1 - \varepsilon}{\varepsilon} \leq \left[ \frac{1 + \rho \left( 1 - 2p_v^{(l)} \right)}{1 - \rho \left( 1 - 2p_v^{(l)} \right)} \right]^{2b_j - j + 1}, \quad (26) \]

where we note that \( 2b_j - j + 1 = b_j - (j - 1 - b_j) = \Delta b \) is the difference between the check nodes that disagree with the variable node and those that do not. Equation (26) therefore states that only this difference matters and needs to be optimized.

Designing irregular graphs and numerically analyzing their thresholds reveals that better codes can be found by allowing the node degrees to vary. In [22], the authors present the improved irregular codes shown in Table 1.

2.4 The AWGN Channel

The additive white Gaussian noise channel discussed in [30, Chapter 2], is probably the most important representative of practical channels, especially for wireless communications. In this section, we apply the method of density evolution to the AWGN channel. An exact solution for its density evolution is possible, but involved. Furthermore, the inevitable numerical treatment of probability densities gives little insight into the process [28]. A close approximation can be found with less effort if messages in the decoder are assumed to have a Gaussian density. This is in general not the case, but the Gaussian approximation greatly reduces the complexity and gives results which are very close to the exact solutions [8].

As in the rest of the chapter, we assume that binary transmission is used with unit-energy symbols from the alphabet \{-1, +1\}. We can also assume, without loss of generality, that the transmitted message consists only of +1s corresponding to the all-zero codeword, since the LDPC code is linear. A correct message bit in the decoder is therefore one with a positive sign, and an incorrect message is one with a negative sign. The probability density function of log-likelihood messages at the output of a matched-filter receiver is

\[ f_Y(y) = \sqrt{\frac{N_0}{16\pi}} \exp \left( -\frac{N_0}{16} \left( y - \frac{4}{N_0} \right)^2 \right), \quad (27) \]
Table 1: Some irregular LDPC codes which outperform regular Gallager codes on the BSC at rate $R = 0.5$. The capacity threshold is at 0.111.

<table>
<thead>
<tr>
<th>Code</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_3$</td>
<td></td>
<td>.123397</td>
<td>.093368</td>
<td></td>
</tr>
<tr>
<td>$\lambda_4$</td>
<td></td>
<td>.555093</td>
<td>.346966</td>
<td></td>
</tr>
<tr>
<td>$\lambda_5$</td>
<td>.496041</td>
<td>.284961</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_6$</td>
<td>.173862</td>
<td>.124061</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{16}$</td>
<td></td>
<td></td>
<td>.321510</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{21}$</td>
<td>.077225</td>
<td></td>
<td>.159355</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{23}$</td>
<td></td>
<td>.252871</td>
<td></td>
<td>.400312</td>
</tr>
<tr>
<td>$\lambda_{27}$</td>
<td></td>
<td>.068844</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{29}$</td>
<td></td>
<td>.109202</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{30}$</td>
<td></td>
<td>.119796</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{100}$</td>
<td></td>
<td>.293135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho_{14}$</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho_{22}$</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\varepsilon^*$</td>
<td>.0505</td>
<td>.0533</td>
<td>.0578</td>
<td>.0627</td>
</tr>
</tbody>
</table>

as can easily be calculated from the input signal $y \sim \mathcal{N}(1, N_0/2)$. This log-likelihood density fulfills what is known as the symmetry conditions

$$f_Y(-y) = f_Y(y)e^{-y}$$

$$m_Y = \frac{\sigma^2_Y}{2}$$

where $m_Y$ and $\sigma_Y^2$ are the mean and variance of $Y$, respectively.

Condition (29), also called the consistency condition, implies that the density $f_Y$ is completely determined by the mean $m_Y$. The consistent Gaussian assumption allows us to consider the evolution of a single parameter.

Since the output message is the sum of incoming messages (6), and noting that messages are assumed to be independent due to the local tree structure assumption, we obtain for variable nodes under probability propagation in the log domain

$$m_v^{(l)} = m_v^{(0)} + (d_v - 1)m_u^{(l-1)},$$

where $m_v^{(l)}$ is the mean output from the variable node at iteration $l$, $m_u^{(l-1)}$ is the mean output from a check node at iteration $(l - 1)$, and $m_v^{(0)}$ is equal to the mean of $f_Y(y)$ in (27), which equals $\frac{4}{N_0}$.

The density evolution through a check node according to (5) is more complicated. To find the evolution of the mean, we assume the messages are i.i.d. and, after moving the $\tanh(\cdot)^{-1}$ over to the left side, take the expectation on both sides to obtain

$$E \left[ \tanh \left( \frac{U}{2} \right) \right] = E \left[ \tanh \left( \frac{V}{2} \right) \right]^{d_v-1}. \quad (31)$$
To simplify our analysis, we define a function $\phi$ as

$$
\phi(m_u) = 1 - E\left[\tanh\left(\frac{U}{2}\right)\right] \\
= 1 - \frac{1}{4\pi m_u} \int_{\mathbb{R}} \tanh\left(\frac{u}{2}\right) \exp\left[-\frac{1}{4m_u} (u - m_u)^2\right] du.
$$

(32)

where $m_u > 0$. The motivation for defining $\phi(\cdot)$ is mainly computational. The bulk of the computational difficulty occurs in the integral of (32). We will see that there are some convenient approximations to $\phi(\cdot)$ which can greatly speed computations with only minor effects on accuracy.

Note that

$$
\tilde{u} = E\left[\tanh\left(\frac{U}{2}\right)\right] = \Pr(U = 1) - \Pr(U = -1)
$$

(33)

can be interpreted as a soft bit decision on $u$.

The mean of the check node’s output message in a regular LDPC code is therefore

$$
\tilde{u}^{(l-1)} = \left(\tilde{v}^{(l-1)}\right)^{d_c-1} \\
m_u^{(l-1)} = \phi^{-1}\left(1 - \left[1 - \phi\left(m_u^{(l-1)}\right)\right]^{d_c-1}\right).
$$

(34)

The results (30) and (34) are sufficient to compute thresholds for regular ensembles. A code converges if $m^{(l)} \to \infty$ as $l \to \infty$. An example of the behavior of this mean value for a regular (4,8) LDPC code is shown in Figure 12 below, showing that convergence of the code (divergence of (34)) occurs for $E_b/N_0 > 1.55\text{dB}$.

![Figure 12: Check node mean evolution for a regular (4,8) LDPC code as a function of the iteration $l$.](image)

The exact computation of $\phi(\cdot)$ and $\phi^{-1}(\cdot)$ is computationally expensive and slow. The computation speed can be greatly improved using a few approximations. For small $x$, perhaps a good approximation for $\phi$ is [8]:

$$
\phi(x) \approx e^{ax^2 + \beta},
$$

(35)
where \( \alpha = -0.4527, \beta = 0.0218, \) and \( \gamma = 0.86. \) For larger \( x, \) the following upper and lower bounds become tight, so that their average can be used as a good approximation to \( \phi: \)

\[
\sqrt{\frac{\pi}{x}} e^{-\frac{3}{x}} \left( 1 - \frac{3}{x} \right) < \phi(x) < \sqrt{\frac{\pi}{x}} e^{-\frac{4}{x}} \left( 1 - \frac{1}{7x} \right). \quad (36)
\]

The figure below shows a numerical comparison between the approximation
\[
\phi(m) \approx \begin{cases} 
\exp (-0.4527m^{0.86} + 0.0218) ; & \text{for } m < 19.89 \\
\sqrt{\frac{\pi}{m}} \exp \left( -\frac{\pi}{4m} \right) \left( 1 - \frac{11}{m} \right) ; & \text{for } m \geq 19.89 
\end{cases} \quad (37)
\]
and the actual function value. Since the difference is indistinguishable, it is plotted separately also.

![Figure 13: Exact function \( \phi(m) \) and its approximation (37).](image)

The density evolution for irregular ensembles is found in analogy with the derivation of (10) and the treatise above, but requires some modifications. We treat node degrees as random variables. The densities of the messages entering a check node are now Gaussian mixtures, with (30) and (34) as partial solutions, due to the fact that different degree variable nodes generate these messages.

More precisely, the mean value of the messages leaving a degree-\( i \) variable node is given by
\[
m_{v,i}^{(l-1)} = (i-1)m_{u}^{(l-1)} + m_{v}^{(0)}. \quad (38)
\]

In turn, the check node output signal for a node of degree \( j \) obeys
\[
E \left[ \tanh \left( \frac{U}{2} \right) \right] = \prod_{i=1}^{j-1} E \left[ \tanh \left( \frac{V_i}{2} \right) \right] \quad (39)
\]
due to the independence of messages entering the check node. We now obtain
\[
\phi \left( m_{u,j}^{(l)} \right) = 1 - \prod_{i=1}^{j-1} \sum_{i=1}^{d_v} \lambda_i \left( 1 - \phi \left( m_{v,i}^{(l-1)} \right) \right) \\
= 1 - \left[ 1 - \sum_{i=1}^{d_v} \lambda_i \phi \left( (i-1)m_{u}^{(l-1)} + m_{v}^{(0)} \right) \right]^{j-1}. \quad (40)
\]
To obtain the *average* check node output signal, we average (40) over the check node degrees to obtain

\[
m^{(l)}_u = \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=1}^{d_v} \lambda_i \phi \left( (i-1)m^{(l-1)}_u + m^{(0)}_v \right) \right]^{j-1} \right) . \tag{41}
\]

This is the recursive formula for the evolution of the check node mean \( m_u \). Note that the check node output signal may not be exactly Gaussian, but these signals are mixed by the additive variable node which produces a Gaussian output with high accuracy, especially if \( d_v \) is large, and hence working with the single mean (41) assuming the signal to be Gaussian produces results that are quite accurate.

With these preliminaries, we now proceed to determine the thresholds for the AWGN channel, which will inform us about the potential of LDPC codes with message passing decoding. From our assumption that the all-zero codeword is transmitted, correct bits and messages have positive signs. Decoding is error-free if the mean \( m^{(l)}_v \) in (38) diverges to infinity as the number of iterations becomes large, since an LLR value distributed according to (27) becomes unequivocal as \( m_Y \to \infty \). In this case, only messages with positive sign can occur, and the probability of an incorrect message goes to zero.

The approximate threshold for the AWGN is the boundary of the set of parameters \( m^{(0)}_v = 4/N_0 \) for which \( m^{(l)}_v \to \infty \) as \( l \to \infty \). It is numerically more convenient to find solutions for a recursion which converges to zero. (38) and (41) can be rearranged to produce the alternate expressions. In particular, instead of considering \( m \), we consider the evolution of \( \phi(m) \) (see Figure 13). If we define

\[
r^{(l)} = \sum_{i=1}^{d_v} \lambda_i \phi \left( m^{(l)}_{v,i} \right) \tag{42}
\]

then

\[
m^{(l)}_u = \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left( 1 - \left[ 1 - r^{(l-1)} \right]^{j-1} \right) \tag{43}
\]

from (41). Combining

\[
\phi \left( m^{(l)}_{v,i} \right) = \phi \left( s + (i-1)m^{(l)}_u \right) , \tag{44}
\]

with (43) substituted into (42) we finally obtain

\[
r^{(l)}(s) = \sum_{i=1}^{d_v} \lambda_i \phi \left[ s + (i-1) \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left( 1 - \left[ 1 - r^{(l-1)}(s) \right]^{j-1} \right) \right] \overset{\text{def}}{=} h(r, s) , \tag{45}
\]

where we set \( s = m^{(0)}_v = 4/N_0 \) as the initial channel LLR. Decoding is error-free if the recursion \( r^{(l)} \) converges to zero as \( l \to \infty \). This condition is satisfied if and only if \( r > h(s, r) \) for all \( r \in (0, \phi(s)) \). The threshold is therefore identified as

\[
s^* = \inf \left\{ s \in \mathcal{R}^+ : h(s, r) - r < 0, \forall r \in (0, \phi(s)) \right\} . \tag{46}
\]

Using (46), the threshold can be found numerically using a simple search algorithm. It is common to express this threshold in terms of the standard deviation of the channel noise, \( \sigma^* = \sqrt{2/\pi} \). Calculation of the inverse \( \phi^{-1} \) is done numerically since no simple analytical solution is known. However, the inverse of the approximation to \( \phi \) is easily computed.

The function \( h(s, r) \) can also be studied graphically as shown in Figure 14 to obtain a visual impression of the rate of convergence. If \( h(s, r) - r \) is close to zero, then a “slow region” or “bottleneck”
occurs. If the channel parameter places $r_0$ in or near the bottleneck region, then many iterations may be required to achieve good performance. The bottleneck tends to be more severe when $s$ is very close to the channel’s threshold.

The iterated behavior of $h(s, r)$ is illustrated in Figure 14. The curves shown are for the $d_v = 15$ degree distribution reported in Table 2. The curve for $\sigma = .9440$ is at the threshold for this degree distribution. We see that $h(s, r) \to 0$ as the number of iterations goes to infinity. The “channel” which exists between $h(s, r)$ and $r$ is very narrow, especially for small $r$. This means that many iterations are required to achieve the limiting performance. As the noise standard deviation $\sigma$ decreases, the channel becomes more open and fewer iterations are required for convergence.

![Figure 14: Iterative Behavior of the function $h(s, r)$, illustrating the convergence of LDPC codes on AWGN channels.](image)

Table 2 show degree distributions and thresholds for rate $R = 1/2$ irregular LDPC codes for various maximal variable node degrees. The channel capacity limit at this rate occurs for $\sigma = 0.9787$, illustrating the excellent performance of these codes. The thresholds are computed according to both the exact [28] and the Gaussian approximative density evolution [8].

The accuracy of density evolution using the Gaussian approximation follows different trends for regular and irregular codes. For regular codes, the accuracy has been shown to improve with increasing $d_v$. For irregular codes, the accuracy is worse when the maximum variable node degree is increased [8]. It is believed that the approximation is more accurate for regular codes, and that irregular codes with maximal variable node degree $d_v$ less than about ten are “regular enough” for an accurate approximation.

Other results from [8] are useful for optimizing LDPC codes, and will be summarized here. One important result is the concentration theorem for check node degree distributions. A concentrated degree distribution satisfies

$$
\rho(x) = \rho_k x^{k-1} + (1 - \rho_k) x^k.
$$

A distribution of this form maximizes the rate of convergence under decoding iterations.

Based on the concentration theorem, Chung et. al. [7] designed LDPC codes for the AWGN channel using a discretized density evolution, that is, they quantized the probability densities and propagated
Table 2: Exact ($\sigma^*$) and approximate ($\sigma^*_{GA}$) thresholds for various rate $R = \frac{1}{2}$ degree distributions on the AWGN channel. The capacity limit is .9787.
this quantized PDF with accuracies of up to 14 bits. The degree distribution was obtained via linear optimization using constraint rules. The degree distributions of some of the optimized codes are given in Table 3.

The check node distribution is given as $\rho_{av} = (1 - \rho_k)(k - 1) + \rho_k k = k - 1 + \rho_k$ as a single number. Given that the Shannon capacity threshold is at $\sigma = 0.97869$, in terms of signal-to-noise ratio, these codes come to within 0.0247, 0.0147, and 0.0045dB of the Shannon capacity respectively.

Chung et. al. [7] also provide simulation results for blocklengths of $10^7$, which are reproduced in Figure 15, illustrating the tight predictions of the density evolution method.

### 2.5 LDPC Code Design via EXIT Charts

Extrinsic Information Transfer (EXIT) charts were invented by ten Brink [32] and subsequently became very popular in the analysis of turbo codes, both for parallel as well as serial concatenation. EXIT charts are an easy visual way of describing the convergence behavior of iterative coding methods and can also be applied to LDPC codes. In order to do this, we consider the two nodes essentially as two types of codes, the parity check node representing a parity check code, and the variable node representing a repetition code. As we know, messages are exchanged between these two nodes and density evolution is used to track the evolution of the message statistics through the iterations, where

<table>
<thead>
<tr>
<th>$d_v$</th>
<th>100</th>
<th>200</th>
<th>8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_2$</td>
<td>.170031</td>
<td>.153425</td>
<td>.096294</td>
</tr>
<tr>
<td>$\lambda_3$</td>
<td>.160460</td>
<td>.147526</td>
<td>.093393</td>
</tr>
<tr>
<td>$\lambda_6$</td>
<td>.112837</td>
<td>.041539</td>
<td>.033599</td>
</tr>
<tr>
<td>$\lambda_7$</td>
<td>.047489</td>
<td>.147551</td>
<td>.091918</td>
</tr>
<tr>
<td>$\lambda_{10}$</td>
<td>.011481</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{11}$</td>
<td>.091537</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{15}$</td>
<td></td>
<td>.031642</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{18}$</td>
<td></td>
<td>.047938</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{19}$</td>
<td></td>
<td>.119555</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{20}$</td>
<td></td>
<td>.086563</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{26}$</td>
<td></td>
<td>.152978</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{27}$</td>
<td></td>
<td>.036131</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{50}$</td>
<td></td>
<td>.093896</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{55}$</td>
<td></td>
<td>.036379</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{56}$</td>
<td></td>
<td>.126714</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{70}$</td>
<td></td>
<td>.006035</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{100}$</td>
<td></td>
<td>.217056</td>
<td>.018375</td>
</tr>
<tr>
<td>$\lambda_{105}$</td>
<td></td>
<td>.086919</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{200}$</td>
<td></td>
<td>.179373</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{400}$</td>
<td></td>
<td>.089018</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{600}$</td>
<td></td>
<td>.057176</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{2000}$</td>
<td></td>
<td>.085816</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{3000}$</td>
<td></td>
<td>.006163</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{6000}$</td>
<td></td>
<td>.003028</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{8000}$</td>
<td></td>
<td>.118165</td>
<td></td>
</tr>
</tbody>
</table>

$\rho_{av} = k - 1 + \rho$

| $\sigma^*$ | 0.97592 | 0.97794 | 0.9781869 |
| $(E_b/N_0)^*$ [dB] | 0.212 | 0.194 | 0.192 |

Table 3: Parameters of near-capacity optimized LDPC codes.
numerical probability density functions are used in the “exact” analysis, and mean values in the Gaussian approximation. A third possible statistic is the mutual information between the messages and their underlying symbols, i.e., we understand the messages as “noisy” versions of the actual variables, the code bits, and measure their reliability by measuring the mutual information between the bits $x$ and their messages $\mu$ and $\beta$ respectively. This mutual information takes the role of the mean values and is computed both at the input and the output of a node, as shown in Figure 16.
$I_{A,\text{chk}} = I(x; \beta)$, as well as at the output of the nodes as $I_{E,\text{var}}$ and $I_{E,\text{chk}}$, respectively. These values are computed numerically, assuming a Gaussian distribution at the input, but imposing no condition on the output PDF.

Despite these approximations, very precise error cliff onset predictions can be made using EXIT. Figure 17 shows the two EXIT curves for an optimized degree distribution at $E_b/N_0 = 0.7$ dB, which is close to the code’s threshold. The dashed curve is the EXIT curve for the parity-check node, and the solid curve the one for the variable node. Close agreement between the two curves provides a small open channel through which the mutual information parameter can evolve from 0 to the value of 1, which represents convergence.

The code with parameters shown in Figure 17 was designed using EXIT charts [17] to create an irregular LDPC with a low-SNR error cliff code.

![EXIT chart of matched and optimized irregular LDPC codes.](image)

A graphical optimization was used to optimize the behavior of the transfer curves in Figure 17, which yielded the irregular code shown with an error cliff about 0.5 dB better than a regular code of the same rate. Simulation results for both the degree optimized code and a regular (3,6) LDPC code are shown in Figure 18. The code with the error floor is the irregular design. It is clearly evident that the irregular code has a lower SNR error cliff. However, the error floor of the irregular code is significantly higher than the error floor of the regular code. Both codes are optimized in avoiding cycles (see next section), but the avoidance of cycles is substantially more successful for the regular code, where all cycles of length 8 and shorter are avoided, while in the irregular code only cycles of length 6 and shorter were avoided. The regular codes have the steep error cliff, the irregular codes are the ones with the visible error floors. The regular codes lose about 0.5 dB in the error cliff region, as predicted by the analysis.
This leads into the question of code optimality. While the degree optimization can achieve excellent performance in the error cliff region, there is no guarantee that the code will have a low error floor. Indeed, codes with excellent cliff performance seem to be more difficult to optimize in the error floor region as illustrated by the simple cycle-optimized codes in Figure 18. The issue of error floor optimization is picked up and discussed in more detail in the next section.

![Figure 18: Performance results for regular and irregular cycle optimized LDPC codes of rate $R = 1/2$ for a block length of 4000 bits. Dashed lines are for the block or frame error rates, solid lines for the BER.](image)

The issue of regular versus irregular LDPC codes doesn’t stop at the error floor question. As we have observed, for rate $R = 1/2$ codes, substantial gains are provided by an irregular code over a regular code, gains in the order of half a dB or more. For lower rates, even more can be gained, while the difference between regular and irregular coded performance in the waterfall region diminishes as the rate increases. Figure 19 illustrates this point by comparing the thresholds of the best regular LDPC codes to the BPSK Shannon bound, clearly illustrating that high-rate regular LDPC codes are very efficient, with the high-rate regular codes having thresholds a mere 0.5dB away from the BPSK capacity limit.

### 3 The Error Floor Problem

As observed with the original turbo codes, LDPC codes which are constructed randomly suffer from an error floor. For moderate to large-sized LDPCs, this error floor typically sets in around bit error
rates of $P_b = 10^{-5}$ or $10^{-6}$, see Figure 18. Especially for applications that require very low frame error rates, this error floor is a nuisance and must be controlled.

While analytical analysis of the error floor remains an elusive and difficult problem, there are several strategies that have been applied successfully to reducing the error floor. Early on, the existence of short cycles in the graph of the code has been implicated with an elevated error floor, and avoiding such short cycles has proven quite effective. This is known as increasing the girth of the code graph.

However, simply increasing the girth of a code seems to have very limited effect on the error floor, and a number of special construction methods have been proposed with good to very good results, most of which are verified empirically. Furthermore, a tradeoff between low error floors and low SNR thresholds of the codes has been observed as a general rule, that is, codes with very low error floors will perform around half a dB poorer in their convergence thresholds.

### 3.1 Exploiting Variable Node Degrees

One successful method is to realize that not all variable nodes have the same error rates. In fact, due to (38), nodes with a higher degree will have a larger LLR value and thus will be more reliable. We therefore associate nodes of high variable degrees with information nodes, and those of low degrees with

![Figure 19: Comparison of thresholds for regular LPDC codes with the binary BPSK capacity limit.](image-url)
parity nodes. In decoding, we are typically only interested in low error rates among the information
bits, and ignore the parity bits. This is the approach taken in the design of the so-called extended
irregular repeat accumulate codes (eIRA) studied in [36].

The parity check matrix of these codes has a special structure, an example of which is

\[ H = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \]

where the parity section is characterized by the dual diagonal which allows very efficient recursive
encoding, see Section 4.

Yang et al. [36] have constructed eIRA codes using optimized degree profiles. The construction
of the parity-check matrix simply ensured that there were no 4-cycles. The authors noted that if they
biased the distributions such that no degree-3, or no degree-4 information variable nodes existed, they
could trade off the error floor and the convergence threshold. Figure 20 shows the performance of
three such codes of rate \( R = 0.82 \) of size (4161,3430) [36] whose degree distributions are given by

**Code 1:**

\[
\begin{align*}
\lambda(x) &= 0.00007 + 0.1014x + 0.5895x^2 + 0.1829x^6 + 0.1262x^7 \\
\rho(x) &= 0.3037x^{18} + 0.6963x^{19}
\end{align*}
\]

**Code 2:**

\[
\begin{align*}
\lambda(x) &= 0.0000659 + 0.0962x + 0.9037x^3 \\
\rho(x) &= 0.2240x^{19} + 0.7760x^{20}
\end{align*}
\]

**Code 3:**

\[
\begin{align*}
\lambda(x) &= 0.0000537 + 0.0784x + 0.9215x^4 \\
\rho(x) &= 0.5306x^{24} + 0.4694x^{25}
\end{align*}
\]

### 3.2 The ACE Construction

Another approach was proposed in [33], which hypothesized that small sets of variable nodes which
have few connections leaving the set were prone to failure in the iterative decoding process. As a
starting point, we consider stopping sets, whose variables have no connections to the outside. Such
a set is known as having an extrinsic message degree (EMD) of zero. This is illustrated in Figure 21
below. Note that the check nodes may have connections to the outside, but that is irrelevant, since messages through check nodes are weakened by the check node processing function (5), while messages through variable nodes are strengthened, hence the definition of EMD which takes only variable nodes into account:

The extrinsic message degree (EMD) of a set is defined as the number of connections from the variable nodes of the set to outside (check) nodes.

The EMD of a reduced set derived from the stopping set is also shown in Figure 21. It has an EMD of 3. This illustrates the somewhat arbitrary nature of the definition. Nonetheless, this definition has proved quite useful. Note that a stopping set always has an EMD of zero.

Working with the EMD of sets is combinatorially complex, and a computationally more practical measure is needed. Tian et. al. [33] proposed to consider the EMD only of cycles, ignoring possible connection inside the cycle. For example, their definition would ignore the central check node in the right-hand set in Figure 21, and give the associate circle an approximate EMD of 5. In general then, the approximate EMD of a circle of length $2d$ equals

$$ACE = \sum_{i=1}^{d} (\lambda_i - 2).$$ (49)
Smaller Set, EMD = 3
Stopping Set, EMD = 0

Figure 21: Extrinsic message degree of two sets.

On the basis of this *Approximate Cycle EMD*, called the ACE of a cycle, [33] proposed the construction of codes with parameters $d_{ACE}$ and $\eta_{ACE}$, thus defined:

An LDPC code has parameters $(d_{ACE}, \eta_{ACE})$ if all cycles of length $d \leq 2d_{ACE}$ have $ACE \geq \eta_{ACE}$.

This definition opens the construction of LDPCs with many different sets of parameters. That this procedure can produce good codes is illustrated in Figure 22 [33], which compares two codes of rate $R = 0.5$ and length 10K.

## 4 LDPC Encoding

### 4.1 Approximate Triangularization

The sparseness of an LDPC parity-check matrix translates into the efficient graph-based decoding algorithms described above. There is, however, an unfortunate side-effect: encoding of LDPC codes has quadratic complexity with respect to the code length. This has prompted several researchers to explore efficient encoding algorithms for LDPC codes. Some of these approaches require specialized code construction.

In general, a linear block code may be encoded by finding a generator matrix, defined by the relation $GH^T = 0$ as discussed earlier. $G$ provides a basis for the code $C$, and encoding of an information sequence $u$ is accomplished by the multiplication $u^T G$. If we assume the code is systematic, then we may also encode directly using the parity check matrix. Suppose a codeword has the structure $x = [x_u x_p]$, where $x_u$ is the information sequence and $x_p$ is the parity sequence.

We can then split $H$ into $[H_u \mid H_p]$, giving the equation

$$H_p x_p^T = H_u x_u^T \; ;$$

(50)
$x_p^T = H_p^{-1}H_u x_u^T$.  

Note also that $H_u$ and $H_p$ are square matrices. There are a few methods available to make this computation efficient. The authors of [29] suggest placing the matrix in “approximate lower triangular” form, in which the upper right corner is populated with only 0s as shown in Figure 23.

A given parity-check matrix $H$ must be placed into this form using only row and column permutations. The distance $g$ is referred to as the “gap”. The gap can be thought of as a measure of the distance from $H$ to a true lower-triangular matrix. We then split $x_p$ into $[p_1, p_2]$, where $p_1$ has length $g$ and $p_2$ has length $m - g$, and multiply $H$ from the left by

$$
\begin{bmatrix}
I & 0 \\
-ET^{-1} & I,
\end{bmatrix},
$$

(52)

giving

$$
\begin{bmatrix}
A & B \\
C - ET^{-1}A & D - ET^{-1}B & 0
\end{bmatrix}.
$$

(53)

The parity-check rule then gives the following equations

$$
Ax_u^T + B p_1^T + T p_2^T = 0,
$$

$$
(C - ET^{-1}A)x_u^T + (D - ET^{-1}B)p_1^T = 0.
$$

(54)
Now define $P = D - ET^{-1}B$, and assume $\phi$ is non-singular. We then arrive at

$$
\begin{align*}
p_1^T &= P^{-1}(C - ET^{-1}A)x_u^T, \\
p_2^T &= -T^{-1}(Ax_u^T + Bp_1^T).
\end{align*}
$$

A judicious arrangement of these computations gives a sequence of complexity $O(n)$. Only the computation $P^{-1}(Cx_u^T - ET^{-1}Ax_u^T)$ has complexity $O(g^2)$, because $P^{-1}$ is a dense $g \times g$ matrix. The authors of [29] also show that, for sufficiently large $n$, $g \leq O(\sqrt{n})$ with high probability for a randomly chosen LDPC code. This algorithm therefore provides encoding with linear complexity in the code length $n$.

### 4.2 Triangular LDPC Codes

One way around this problem is to enforce a triangular structure of the parity-check matrix and see if LDPC codes with good performance can still be constructed. Due to the correlations enforced by the structural constraint, a random performance analysis is not strictly possible anymore. However, constructing LDPC codes with triangular parity-check matrices does produce codes with equivalent performance to those with completely random parity-check matrices. This is illustrated in Figure 24 for a large code design.

If $H_p$ is lower triangular, then the BEC decoding algorithm can be used to perform iterative encoding in at most $m$ steps. Consider the following simple example:

$$
H = \begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}.
$$

The constraint graph for this matrix is shown in Figure 25.
Figure 24: Performance results for a triangular and a non-triangular (random) LDPC codes of rate $R = 1/3$ for a block length of 36,993 bits.

Figure 25: Constraint graph for the matrix in (56).

The information bits at the top of the graph are initialized with $\pm 1$ and the parity nodes with 0. When the BEC decoding algorithm is applied, the first round of message passing will cause parity-bit 1 to become known. In the second round, a non-zero message from parity-bit 1 allows parity-bit 2 to become known. Similarly, the parity-bit 3 becomes known in the third round. Encoding therefore takes
place in $m = 3$ iterations, in fact, successively going through each row constraint in the parity-check matrix. In other words, the set of parity bit variables is never a stopping set.

This algorithm is convenient and provides the following advantages (i) Encoding complexity is now linear with $m$, (ii) the encoder circuit can possibly share the same silicon area with a decoder. This can be valuable in transceiver systems. Such a shared dual mode encoder/decoder was presented in [16]. The drawback is the added restriction on code structure which could result in limits on decoder performance, even though there does not seem to be much supportive evidence for it. It is also very important to note that if a lower-triangular matrix is obtained by linear combinations from the original parity-check matrix, then it is also possible to encode using the lower-triangular matrix graph while decoding with the original matrix graph.

The authors of [15] found that encoding can also be performed by means of iterated approximation. We wish to solve $H_p x_p^T = H_u x_u^T$ for $x_p^T$. To do so, we first select an initial guess for $x_p$, which we’ll denote by $x_0$. We will then use an iterative method to obtain subsequent approximations, denoted $x_k$. These approximations will converge to $x_p$ using the following iteration:

$$x_{k+1}^T = (H_p + I) x_k^T + H_u x_u^T,$$  \hspace{1cm} (57)

provided $H_p$ is non-singular. Correct encoding results after $k' \geq k$ iterations if and only if $(H_p + I)^k = 0$. Let $b = H_u x_u^T$. To apply the algorithm, we can split the code’s graph into two parts as shown in Figure 26 – an information-bit section and a parity-bit section. The arrows in the graph indicate which edges will be enabled to transmit edges from check nodes in the encoding algorithm.

![Figure 26: Splitting the code graph into two parts to apply iterative encoding.](image)

The top half of the graph computes $b$. The bottom part only represents the constraint in $H_p$. $H_p$ is an $m \times m$ matrix, so the number of parity nodes (labeled $v_i$) is the same as the number of check nodes (labeled $c_i$) for this part of the graph. The encoding algorithm can then be implemented graphically using the following algorithm:

**Step 1:** Set all $v_i = +1$. Initialize information nodes to $\pm 1$ as appropriate, and propagate messages to yield $b_i$. 


Step 2: Variable nodes send $\mu_{ij} = b_i$ to all $j \in V_i \setminus i$.

Step 3: Check nodes answer with $\beta_{jj} = \prod_{l \in C_j \setminus j} \mu_{lj}$, sent to $v_j$ only.

Step 4: Variable nodes update $v_i = \beta_{ii}$. Return to Step 2 and repeat for the prescribed number of iterations.

This result allows us to construct a code which is guaranteed to be graphically encodable in a fixed number of iterations.

In [15] a method is presented for constructing “reversible” LDPC codes, which can be encoded in two iterations. It is first noted that $(H_p \oplus I)^2 = I \rightarrow H_p^2 = I$. A reversible code may then be constructed by selecting a $2 \times 2$ matrix $A_0$ for which $A_0^2 = I$. $H_p$ can be obtained by recursive application of one of the following rules:

$$A_k = \begin{bmatrix} A_{k-1} & I \\ 0 & A_{k-1} \end{bmatrix} \quad \text{or} \quad A_k = \begin{bmatrix} A_{k-1} & 0 \\ 0 & A_{k-1} \end{bmatrix}.$$  \hspace{1cm} (58)

As an example, we can construct the matrix

$$A_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow H_p = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$  \hspace{1cm} (59)

and concatenate it with

$$H_u = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$  \hspace{1cm} (60)

The resulting code is an [8,4] extended Hamming code, whose graph is shown in Figure 27.

This graph can be used for decoding as well as encoding. It may also be used to encode with the earlier BEC algorithm if the non-arrowed message paths are switched off during encoding. In summary then, encoding of LDPC codes can be achieved with linear complexity.

5 Repeat Accumulate Codes

The near-capacity performance of low density parity check codes and the elegance of the graphical description of the codes and the decoding algorithm led to the investigation of graph-based coding schemes. In this section, we consider a particularly simple variant of LDPC codes known as repeat accumulate (RA) codes [11]. RA codes are serially concatenated codes, and are covered from that perspective in ([30, Chapter 11]). They have a convenient graphical representation that enables them to benefit from the notion of irregularity and to be analyzed using density evolution.

A block diagram of a nonsystematic repeat accumulate encoder is shown in Figure 28. The outer encoder is a simple repetition code with rate $R = 1/q$ and the inner encoder is a rate one recursive encoder with feedback polynomial $1 + D$, that is, an accumulator. The inner and outer encoders are separated by an interleaver or permuter and the overall code rate is $R = 1/q$. It is worth noting at this point that an obvious advantage of this encoder is its simplicity. The simplicity of these codes is deceiving. For large block lengths and rates $R = 1/q \leq 1/3$, these codes perform within roughly 1 dB of the Shannon limit on the AWGN channel [11].
The graphical representation of the RA code of Figure 28 is straightforward and best illustrated by an example. Let \( q = 3 \) and the information block length \( k = 2 \), then the graph of the resulting code is shown in Figure 29, illustrating again the close relationship between RA and LDPC codes, see also (48). From this graph, it is clear that each variable node has degree \( q = 3 \) and that each parity node has degree 2 due to the accumulator. Thus, the graph of an RA code is a regular bipartite graph with degree \( q \) information variable nodes, degree 1 check nodes and degree 2 parity nodes.

It is evident that RA codes can also be made irregular, this was first suggested in [19]. That is, the number of times that an information bit is repeated is varied. The resulting graph is depicted in Figure 30 and is parameterized by the degree distribution \((\lambda_1, \lambda_2, \ldots, \lambda_J)\) of the information variable nodes and the left degree \( a \) of the intermediate check nodes. Note that in the graph of a regular RA code, such as that depicted in Figure 29, the value of \( a = 1 \).

In [19], the authors considered the subclass of systematic irregular RA codes with a fixed check node left degree of \( a \). In a systematic RA code the graph is used to compute the parity bits and the codeword is \( x = (u_1, u_2, \ldots, u_k, x_1, x_2, \ldots, x_r) \) and the overall code rate is \( R = k/(k+r) \). Using linear programming techniques, they obtained degree sequences for systematic irregular RA codes and then computed exact and approximate thresholds. The results for codes of rate \( R \approx 1/3 \) are shown in Table 4. These codes come within approximately 0.1 dB of the capacity limit.
Check Nodes
Parity Nodes (Codeword Bits)

Figure 29: Tanner graph of an RA code with $q = 3$ and $k = 2$.

$\lambda_1$ $\lambda_2$ $\lambda_J$

Interleaver

Check Nodes
Parity Nodes (Codeword Bits)

Figure 30: Tanner graph of an irregular RA code.

References


<table>
<thead>
<tr>
<th>$a$</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_2$</td>
<td>.139025</td>
<td>.078194</td>
<td>.054485</td>
</tr>
<tr>
<td>$\lambda_3$</td>
<td>.222155</td>
<td>.128086</td>
<td>.104315</td>
</tr>
<tr>
<td>$\lambda_5$</td>
<td>.160813</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_6$</td>
<td>.638820</td>
<td>.036178</td>
<td>.126755</td>
</tr>
<tr>
<td>$\lambda_{10}$</td>
<td></td>
<td>.229816</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{11}$</td>
<td></td>
<td>.016484</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{12}$</td>
<td>.108828</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{13}$</td>
<td>.487902</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_27$</td>
<td></td>
<td>.450302</td>
<td></td>
</tr>
<tr>
<td>$\lambda_28$</td>
<td></td>
<td>.017842</td>
<td></td>
</tr>
<tr>
<td>Rate</td>
<td>0.333364</td>
<td>0.333223</td>
<td>0.333218</td>
</tr>
<tr>
<td>$\sigma^+$</td>
<td>1.1981</td>
<td>1.2607</td>
<td>1.2780</td>
</tr>
<tr>
<td>$(E_b/N_0)^* [\text{dB}]$</td>
<td>0.191</td>
<td>-0.250</td>
<td>-0.368</td>
</tr>
<tr>
<td>$\sigma_{GA}$</td>
<td>1.1840</td>
<td>1.2415</td>
<td>1.2615</td>
</tr>
<tr>
<td>$(E_b/N_0) [\text{dB}]$</td>
<td>0.293</td>
<td>-0.117</td>
<td>-0.255</td>
</tr>
<tr>
<td>Capacity (dB)</td>
<td>-0.4953</td>
<td>-0.4958</td>
<td>-0.4958</td>
</tr>
</tbody>
</table>

Table 4: Degree profiles for optimized systematic irregular repeat accumulate codes [19].


