AN INTRODUCTION TO LOW-DENSITY PARITY-CHECK CODES

Todd K. Moon, Jacob H. Gunther
Electrical and Computer Engineering Department
Utah State University, Logan, UT 84322-4120
Todd.Moon, jake@ece.usu.edu

ABSTRACT

Low-Density Parity-Check (LDPC) codes are powerful codes capable of nearly achieving the Shannon channel capacity. This paper presents a tutorial introduction to LDPC codes, with a detailed description of the decoding algorithm. The algorithm propagates information about bit and check probabilities through a tree obtained from the Tanner graph for the code. This paper may be useful as a supplement in a course on error-control coding or digital communication.

KEYWORDS

Error correction coding, iterative decoding, channel capacity, parity-check codes

Introduction

In 1948, Claude Shannon established the concept of the capacity of a channel, which is the maximum rate of transmission for reliable communication over the channel, and proved the existence of a sequence of error-correcting codes that can achieve arbitrarily small probability of error as the block length of the code increases [1]. Since that time, establishing a communications system with a rate approaching the channel capacity that can be encoded and decoded with reasonable complexity has been a holy grail of engineers and mathematicians. The end of the search was (nearly) declared in 1993 with the introduction of a family of codes known as turbo codes [2]. Using an iterative decoding algorithm, instances of these codes are able to reach within a fraction of a dB of channel capacity in an additive white Gaussian noise (AWGN) channel.

Since the introduction of turbo codes, researchers have rediscovered a family of codes proposed in 1962 by Robert Gallager [3, 4], the low-density parity-check codes (sometimes called Gallager codes [5]), which have performance exceeding, in some cases, that of turbo codes (see, e.g., [6] and [7]), with iterative decoding algorithms which are easy to implement (with the per-iteration complexity much lower than the per-iteration complexity of turbo decoders) and are also parallelizable in hardware [8]. There are other potential advantages to LDPC codes as well. In a very natural way, the decoder declares a decoding failure when it is unable to correctly decode, whereas turbo decoders must perform extra computations to determine if the algorithm has converged [9, 10, 11]. Also, LDPC codes of almost any rate and blocklength can be created simply by specifying the

shape of the parity check matrix, while the rate of turbo codes is governed largely by a puncturing schedule, so flexibility in rate is obtained only through considerable design effort. Unlike turbo codes, which can exhibit a noise floor phenomenon due to low-weight codewords [12], LDPC codes do not exhibit a noise floor [3]. As an additional boon, LDPC codes are not patent protected [13]. On the negative side, LDPC codes have a significantly higher *encode* complexity than turbo codes, and decoding may require many more iterations than turbo decoding (with implications for latency).

It is a curious twist of history that LDPC codes, which are among "perhaps the best error-correcting codes in the world" [13] should have been largely unnoticed for so long, and in fact, have occasionally been perceived to be "bad" codes [14, p. 114].[1] Among the reasons that the codes might have been overlooked are that contemporary investigations in concatenated coding overshadowed LDPC codes, and the hardware of the time could not support effective decoder implementations. As a result, LDPC codes remained largely unstudied for over thirty years, with only scattered references to them appearing in the literature, such as [15, 16, 17]. Recently, however, they have been strongly promoted, beginning with the work of MacKay [5, 18, 13, 19, 7, 20, 21]. Because LDPC codes have iterative decoding algorithms, they fall in the purview of graph-based algorithms for decoding; see, e.g., [8, 11, 22, 23]. Both historically and recently, LDPC codes have been proven to be capable of closely approaching the channel capacity. In fact, the proof of the distance properties of LDPC codes demonstrates such strong performance for these codes that it has been termed a "semiconstructive proof of Shannon's noisy channel coding theorem" [5, p. 400].

The study of LDPC codes provides an introduction to many of the themes of modern error-control coding: block codes described by matrices, graphs associated with a code, trellis-based algorithms, soft decision decoding, and iterative decoding. Given the strength of LDPC codes, it seems important for students to learn about them — they will be used in forthcoming communications and data storage systems. Given the variety of concepts necessary to understand the decoding, it seems pedagogically valuable — though challenging — to present the material to students.

Because of the "novelty" of these rediscovered codes, it is not surprising that current digital communications textbooks do not discuss them. Recent books on error-control coding [24, 25] briefly present them, but the presentations are too brief to explain the decoding algorithm. Furthermore, the available literature on LDPC codes presumes a fairly high level of background. The intent of this paper is to provide an introductory exposition of LDPC codes, with particular focus on clarifying the decoding algorithm. The paper could be used, for example, as a supplement to a course in error-control coding.

Despite their relative novelty, work is underway in a variety of laboratories to implement LDPC codes in DSL, wireless links, and high density magentic recording applications, as well as applications to space-time coding.

[1] In fairness, it should be pointed out that in [5] and on the web version of the paper, the author retracts his denigration of the LDPC codes, recognizing that they are "very good" codes.

Coding theory background

Error-correction codes are a way of introducing redundancy into a data stream in a precisely-defined manner, so that errors that occur in the data stream can be corrected. For $N > K$, an (N, K) block coder partitions a data stream into successive blocks of K bits, to which $N - K$ additional bits are adjoined; these additional bits are called the parity bits.[2] The block of N bits is called the codeword and the quantity $N - K$ is the redundancy of the code. One measure of the utility of a code is the number of errors that it can correct. The *rate* of a code is the ratio of the number of input bits to the number of output bits, $R = K/N$. It is typically desirable to have codes with good error-correction capability and the highest possible rate (or smallest redundancy).

An important class of block codes is the linear block codes, which can be characterized by two matrices, the generator matrix and the parity check matrix. The generator matrix of a linear block code is an $N \times K$ matrix[3] G. The code is the span of the columns of G. That is, a vector $\mathbf{c}$ of length N is a codeword if and only if

$$\mathbf{c} = G\mathbf{m} \tag{1}$$

(operations modulo 2) for some "message" vector $\mathbf{m}$ of length K. When the generator matrix contains a $K \times K$ identity, so that

$$G = \begin{bmatrix} P \\ I \end{bmatrix},$$

then the message vector $\mathbf{m}$ appears explicitly as part of the codeword $\mathbf{c}$; such a generator is said to be in systematic form.

The parity check matrix A of a linear block code is an $M \times N$ matrix, with $M < N$, with the property that $\mathbf{c}$ is a codeword in the code if and only if

$$A\mathbf{c} = \mathbf{0}. \tag{2}$$

Writing

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_M^T \end{bmatrix},$$

we see that $\mathbf{c}$ is a codeword if and only if $\mathbf{a}_m^T\mathbf{c} = 0$ for $m = 1, 2, \ldots, M$. The equation $\mathbf{a}_i^T\mathbf{c} = 0$ is said to be a linear parity-check constraint on the codeword $\mathbf{c}$. We will use the notation $z_m = \mathbf{a}_m^T\mathbf{c}$, and call z_m a parity check, or, more simply, a check. By (1) and (2), $AG = \mathbf{0}$.

---

[2] While binary codes are the only codes discussed in this paper, it is straightforward to generalize to codes with larger symbol sets.

[3] In most of the error-control coding literature, G is taken to be a $K \times N$ matrix, and message vectors and code vectors are taken to be row vectors, so that $\mathbf{c} = \mathbf{m}G$. However, in the literature of low-density parity-check codes, the convention is to use column vectors, which is followed in this paper.

Example 1 *Let*

$$A = \begin{bmatrix} \mathit{1} & \mathit{1} & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ \mathit{1} & \mathit{1} & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3}$$

be a binary parity check matrix. Then, for example, for a codeword $\mathbf{c} = [c_1, c_2, \ldots, c_{10}]$*, we must have*

$$c_1 + c_2 + c_3 + c_6 + c_7 + c_{10} = 0$$

$$c_1 + c_3 + c_5 + c_6 + c_8 + c_9 = 0$$

etc. Thus, bits c_1, c_2, c_3, c_6, c_7 *and* c_{10} *participate in check* z_1*. Also, from* A *it may be observed that bit* c_1 *is involved in checks* z_1, z_2 *and* z_5*. (The italicized elements of* A *are discussed below.)*

A *parity check code* of length N is a linear block code whose codewords all satisfy a set of M linear parity check constraints [26, 27]. That is, the code can be specified by an $M \times N$ parity check matrix A. Each linearly independent constraint reduces the number of valid codewords in the code by a factor of one half. Let $r = \mathrm{rank}(A) \leq M$ be the number of linearly independent constraints; then the dimension of the code is $K = N - r$, and each codeword can represent K information bits. The code rate is thus $R = K/N$.

For a code specified by a parity check matrix A, it is expedient for encoding purposes to determine the corresponding generator matrix G. A systematic generator matrix may be found as follows. Using Gaussian elimination with column pivoting as necessary (with binary arithmetic) determine an $M \times M$ matrix A_p^{-1} so that

$$H = A_p^{-1} A = \begin{bmatrix} I & A_2 \end{bmatrix}.$$

(If such a matrix A_p does not exist, then A is rank deficient, $r = \mathrm{rank}(A) < M$. In this case, form H by truncating the linearly dependent rows from $A_p^{-1} A$. The corresponding code has $R = K/N > (N - M)/N$, so it is a higher rate — better — code than the dimensions of A would suggest.) Having found H, form

$$G = \begin{bmatrix} A_2 \\ I \end{bmatrix}.$$

Then $HG = \mathbf{0}$, so $A_p HG = AG = \mathbf{0}$, so G is a generator matrix for A. While A may be sparse (as discussed below), neither the systematic generator G nor H will necessarily be sparse.

Associated with a parity check matrix A is a graph called the Tanner graph [15] containing two sets of nodes. The first set consists of N nodes which represent the N bits of a codeword; nodes in this set are called "bit" nodes. The second set consists of M nodes, called "check" nodes, representing the parity constraints. The graph has an edge between the nth bit node and the mth check node if and only if nth bit is involved in the mth check, that is, if $A_{m,n} = 1$. Thus, the Tanner graph is a graphical depiction of the parity check matrix. Figure 1 illustrates the graph for A from example 1. A graph such as this consisting of two distinct sets of nodes and having edges only between the nodes in different sets is called a *bipartite* graph. The Tanner graph will be used below to develop

insight which will be used to develop the decoding algorithm. It should be pointed out that the Tanner graph exists for every linear block code, so the decoding algorithm described here applies not just to LDPC codes.

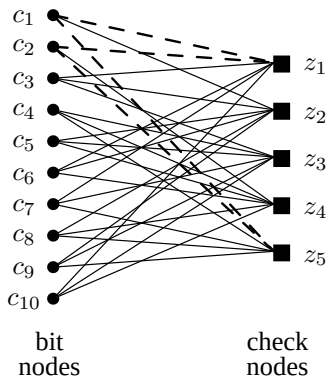


Figure 1: Bipartite graph associated with the parity check matrix A. (The dashed edged correspond to a cycle of length four, as discussed below.)

LDPC codes: construction and notation

A matrix is said to be *sparse* if fewer than half of the elements are nonzero. A LDPC code is simply a code which has a very sparse parity check matrix [5].

The *weight* of a binary vector is the number of nonzero elements in it. The *column weight* of a column of a matrix is the weight of the column; similarly for *row weight*. A LDPC is *regular* if the column weights are all the same and the row weights are all the same. To generate a regular LDPC code, a column weight w_c is selected (typically a small integer such as $w_c = 3$) and values for N (the block length) and M (the redundancy) are selected. Then an $M \times N$ matrix A is generated, typically *at random*, which has weight w_c in each column[4] and row weight w_r in each row. Attaining uniform row weight w_r requires that $w_c N = w_r M$. This structure says that every bit participates in w_c checks, and each check involves w_r bits.

Example 2 *The parity check matrix A of (3) has column weight $w_c = 3$. Strictly speaking, A is not a sparse matrix, since more than half of its elements are nonzero. However, it was chosen as a small matrix to illustrate the concept of a random matrix with fixed-weight columns. Space considerations preclude explicit presentation of more realistic parity check matrices.*

In many LDPC codes, N is taken to be quite large (such as $N > 10,000$), while the column weight is held at around 3 or 4, so the density of 1s in the matrix is quite low. (A column weight of

[4]Actually, it is not necessary for an LDPC code to have equal column weight. In fact, it has been shown [28] that irregular codes having varying column weights can achieve superior performance to the regular codes described here. However, for simplicity of exposition, attention is restricted in this paper to regular codes.

$w_c = 2$ has been found to be ineffective [5].) For the Tanner graph representation of the parity check matrix, each bit node is adjacent to w_c check nodes, and each check node is adjacent to w_r bit nodes.

Since the A matrix is sparse, it can be represented efficiently using lists of its nonzero locations. This notation is used in practical software implementations, as well as to describe the decoding algorithm to be presented below. In this notation, bits are typically indexed by n or n' (e.g., $c_{n'}$) and the checks are typically indexed by m or m' (e.g., z_m). The set of bits that participate in check z_m (that is, the nonzero elements on the mth row of A) is denoted

$$\mathcal{N}_m = \{n : A_{mn} = 1\}.$$

The set of bits that participate in check z_m except for bit n is denoted

$$\mathcal{N}_{m,n} = \mathcal{N}_m \backslash n.$$

The notation $|\mathcal{N}_m|$ indicates the number of elements in the set $\mathcal{N}_m$. These sets should be considered *ordered lists*, with the ith element of $\mathcal{N}_m$ being indicated by $\mathcal{N}_m(i)$.

The set of checks in which bit c_n participates (that is, the nonzero elements of the nth column of A) is denoted

$$\mathcal{M}_n = \{m : A_{mn} = 1\}.$$

For a regular LDPC code, $|\mathcal{M}_n| = w_c$. Let

$$\mathcal{M}_{n,m} = \mathcal{M}_n \backslash m$$

be the set of checks in which bit c_n participates except for check m.

Example 3 *For the parity check matrix of (3),*

$$\mathcal{N}_1 = \{1, 2, 3, 6, 7, 10\}, \quad \mathcal{N}_2 = \{1, 3, 5, 6, 8, 9\}, \quad \textit{etc.}$$

$$\mathcal{M}_1 = \{1, 2, 5\}, \quad \mathcal{M}_2 = \{1, 4, 5\}, \quad \textit{etc.}$$

$$\mathcal{N}_{2,1} = \{3, 5, 6, 8, 9\}, \quad \mathcal{N}_{2,3} = \{1, 5, 6, 8, 9\}, \quad \textit{etc.}$$

$$\mathcal{M}_{2,1} = \{4, 5\}, \quad \mathcal{M}_{2,4} = \{1, 5\}, \quad \textit{etc.}$$

Transmission through a Gaussian channel

The decoding algorithm described below is a soft-decision decoder which makes use of channel information. We will develop the decoder for codewords transmitted through an additive white Gaussian noise (AGWN) channel. When a codeword is transmitted through an AWGN channel, as shown in figure 2, the binary vector $\mathbf{c}$ is first mapped into a transmitted signal vector $\mathbf{t}$. For illustrative purposes, a binary phase-shift keyed (BPSK) signal constellation is employed, so that

the signal $a = \sqrt{E_b}$ represents the bit 1, and the signal $-a$ represents the bit 0. The transmitted signal vector $\mathbf{t}$ has elements $t_n = (2c_n - 1)a$. This signal vector passes through a channel which adds a Gaussian noise vector $\boldsymbol{\nu}$, where each element of $\boldsymbol{\nu}$ is independent, identically distributed with zero mean and variance σ^2. The received signal is

$$\mathbf{y} = \mathbf{t} + \boldsymbol{\nu}. \tag{4}$$

Given the received information, the posterior probability of detection can be computed as

$$\begin{aligned} \Pr(c_n = 1|y_n) &= \frac{p(y_n|t_n = a)\Pr(t_n = a)}{p(y_n)} \\ &= \frac{p(y_n|t_n = a)\Pr(t_n = a)}{p(y_n|t_n = a)\Pr(t_n = a) + p(y_n|t_n = -a)\Pr(t_n = -a)} \\ &= \frac{p(y_n|t_n = a)}{p(y_n|t_n = a) + p(y_n|t_n = -a)} \end{aligned} \tag{5}$$

where it assumed that $\Pr(c_n = 1) = \Pr(c_n = 0) = \frac{1}{2}$. In (5), the notation $\Pr(\cdot)$ indicates a probability mass function, and $p(\cdot)$ indicates a probability density function. For an AWGN channel,

$$p(y_n|t_n = a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y_n - t_n)^2}\Big|_{t_n = a} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y_n - a)^2}. \tag{6}$$

Applying (6) in (5) we obtain

$$\Pr(c_n = 1|y_n) = \frac{1}{1 + e^{-2ay_n/\sigma^2}}. \tag{7}$$

We shall refer to $\Pr(c_n = x|y_n)$, for $x \in \{0, 1\}$, as the *channel posterior* probability, and denote it by $p_n(x)$.

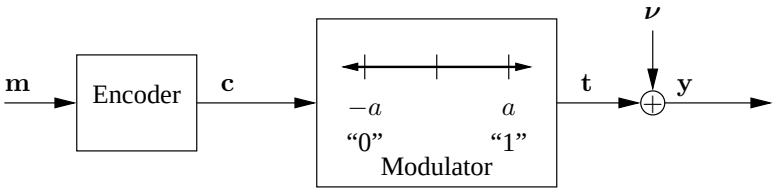


Figure 2: Transmission through a Gaussian channel.

Example 4 *The message vector* $\mathbf{m} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$ *is encoded using a systematic generator* G *derived from (3) to obtain the code vector*

$$\mathbf{c} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T. \tag{8}$$

This is mapped to a signal constellation with $a = 2$ *to obtain the transmitted vector*

$$\mathbf{t} = \begin{bmatrix} -2 & -2 & -2 & 2 & -2 & 2 & -2 & 2 & -2 & 2 \end{bmatrix}^T$$

which is transmitted through an AWGN channel with $\sigma^2 = 2$. The vector

$$\mathbf{y} = \begin{bmatrix} -0.63 & -0.83 & -0.73 & -0.04 & 0.1 & 0.95 & -0.76 & 0.66 & -0.55 & 0.58 \end{bmatrix}^T$$

is received. Using (7), it is found that the channel posterior probabilities are

$$\Pr(\mathbf{c} = 1|\mathbf{y}) = \begin{bmatrix} 0.22 & 0.16 & 0.19 & 0.48 & 0.55 & 0.87 & 0.18 & 0.79 & 0.25 & 0.76 \end{bmatrix}^T \tag{9}$$

If $\mathbf{y}$ were converted to a binary vector by thresholding the probabilities in (9) at 0.5, the estimated vector

$$\begin{bmatrix} 0 & 0 & 0 & \underline{0} & \underline{1} & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

would be obtained. This differs from the original code vector at the two underlined locations. However, note that at the error locations, the channel posterior probability is only slightly different than the threshold 0.5, so that the bits only "weakly" decode to the values 0 and 1, respectively. Other bits more strongly decode to their true values. These weak and strong indications are exploited by a soft-decision decoder.

Decoding parity check codes

The optimal (minimum probability of decoding error) decoder seeks a codeword $\hat{\mathbf{c}}$ which maximizes $\Pr(\mathbf{c}|\mathbf{y}, A\mathbf{c} = \mathbf{0})$, that is, the most probable codeword which satisfies the parity checks, given set of received data $\mathbf{y} = [y_1, y_2, \ldots, y_N]$. However, the decoding complexity for the true optimum decoding of an unstructured (i.e., random) code is exponential in K [29], requiring an exhaustive search over all 2^K codewords, so a modified decoding criterion is employed. Instead, the decoder will attempt to find a codeword having bits c_n which maximize

$$\Pr(c_n|\mathbf{y}, \text{all checks involving bit } c_n \text{ are satisfied}),$$

that is, the posterior probability for a single bit given that only the checks on that bit are satisfied. As it turns out, even this easier, more computationally localized, task cannot be accomplished exactly due to approximations the practical algorithm must make. However, this decoding algorithm has excellent demonstrated performance, and the complexity of the decoding is linear in the code length.

The decoding algorithm deals with two sets of probabilities. The first set of probabilities is related to the decoding criterion, $\Pr(c_n|\mathbf{y}, \text{all checks involving bit } c_n \text{ are satisfied})$. We will denote this by

$$q_n(x) = \Pr(c_n = x|\mathbf{y}, \text{all checks involving } c_n \text{ are satisfied}), \qquad x \in \{0, 1\},$$

or, using the notation defined in section ,

$$q_n(x) = \Pr(c_n = x|\mathbf{y}, \{z_m = 0, m \in \mathcal{M}_n\}), \qquad x \in \{0, 1\}. \tag{10}$$

This probability is referred to as the *pseudoposterior* probability and is ultimately used to make the decisions about the decoded bits. A variant of this probability, called $q_{mn}(x)$, is also used (as seen below) which is

$$q_{mn}(x) = \Pr(c_n = x|\mathbf{y}, \text{all checks, except } z_m, \text{involving } c_n \text{ are satisfied})$$

or, more briefly,

$$q_{mn}(x) = \Pr(c_n = x|\mathbf{y}, \{z_{m'}, m' \in \mathcal{M}_{n,m}\}).$$

The quantities $q_n(x)$ and $q_{mn}(x)$ are probabilities of codeword bits given the observations and the checks. The second set of probabilities employed in the algorithm has to do with the probability of checks given the bits. These indicate the probability that a check is satisfied, given the value of a single bit involved with that check and the observations associated with that check: $\Pr(\text{the } m\text{th check is satisfied}|c_n = x, \mathbf{y})$. This probability is denoted by $r_{mn}(x)$, with $r_{mn}(x) = \Pr(z_m = 0|c_n = x, \mathbf{y})$.

The quantities $q_{mn}(x)$ and $r_{mn}(x)$ are computed only for those values of (m, n) of A that are nonzero. The decoding algorithm incorporates information from the measured data to compute probabilities about the checks, as represented by $r_{mn}(x)$. The information about the checks is then used to find information about the bits, as represented by $q_{mn}(x)$. This, in turn, is used to update the probabilities about the checks, and so forth. Iteration between bit and check probabilities (qs and rs) proceeds until all the parity checks are satisfied simultaneously, or until a specified number of iterations is exceeded. Algorithm 1 is a concise statement of the decoding algorithm; the details are discussed in the sections below. (This particular formulation of the decoding algorithm is due to [5], while concepts from the description are from [4].)

Algorithm 1 Iterative decoding algorithm for binary LDPC codes

---

Input: A, and the channel posterior probabilities $p_n(x) = \Pr(c_n = x|y_n)$, and the maximum # of iterations, L.

Initialization: Set $q_{mn}(x) = p_n(x)$ for all (m, n) with $A(m, n) = 1$.

Horizontal step: For each (m, n) with $A(m, n) = 1$:

Compute $\delta q_{ml} = q_{ml}(0) - q_{ml}(1)$

Compute

$$\delta r_{mn} = \prod_{\{n' \in \mathcal{N}_{m,n}\}} \delta q_{mn'} \tag{11}$$

Compute $r_{mn}(1) = (1 - \delta r_{mn})/2$ and $r_{mn}(0) = (1 + \delta r_{mn})/2$.

Vertical Step: For each (m, n) with $A(m, n) = 1$:

Compute

$$q_{mn}(0) = \alpha_{mn} p_n(0) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(0) \qquad \text{and} \qquad q_{mn}(1) = \alpha_{mn} p_n(1) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(1) \tag{12}$$

where α_{mn} is chosen so $q_{mn}(0) + q_{mn}(1) = 1$.

Also compute the "pseudoposterior" probabilities

$$q_n(0) = \alpha_n p_n(0) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(0) \qquad \text{and} \qquad q_n(1) = \alpha_n p_n(1) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(1)$$

where α_n is chosen so that $q_n(0) + q_n(1) = 1$.

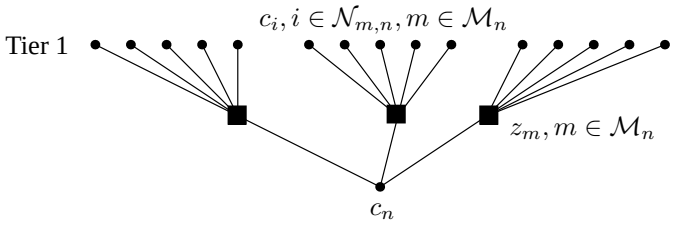
Make a tentative decision: Set $\hat{c}_n = 1$ if $q_n(1) > 0.5$, else set $\hat{c}_n = 0$.

If $A\hat{\mathbf{c}} = 0$, then **Stop**. Otherwise, if #iterations $< L$, loop to **Horizontal Step**
Otherwise, declare a decoding failure and **Stop**.

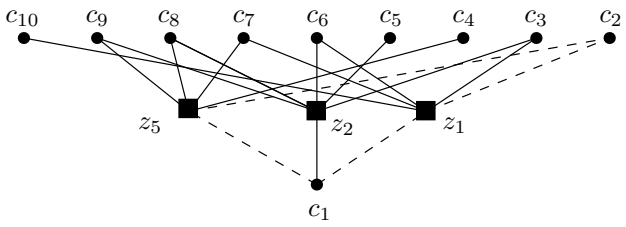
The Vertical Step: Updating $q_{mn}(x)$

We will now explain in detail the decoding algorithm. Consider figure 3(a), which is obtained by selecting an arbitrary bit node c_n from the Tanner graph and using it as the root of a tree, with the subset of the Tanner graph connecting this bit to its checks and the other bits involved in these checks as nodes in the tree. The bits other than c_n which connect to the parity checks are referred to as bits in the *Tier 1* of the tree. We shall assume that the bits represented in the first tier of the tree are distinct, and hence *independent*.

In reality, the portion of the redrawn Tanner graph may not be a tree, since the bits on the first tier may not be distinct. For example, figure 3(b) shows a portion of the actual Tanner graph from figure 1 with the root c_1. In this figure, for example, bit c_2 is checked by both checks z_1 and z_5. There is thus a cycle of length four in the graph, indicated by the dashed lines. This cycle corresponds to the italicized elements of the matrix A in (3). Such a cycle means that the bits in the first tier are not independent (as initially assumed). However, for a sufficiently large code, the probability of such cycles is small (at least for trees represented out to the first tier). We will therefore assume a tree structure as portrayed in figure 3(a), with its corresponding independence assumption.



(a)



(b)

Figure 3: A subset of the Tanner graph. (a) Viewed as a tree, with node for c_n as the root. (b) An actual portion of the Tanner graph, with node for c_1 as root, showing cycle.

Under the assumption of independence of bits in the first tier (no cycles), the checks in the tree are statistically independent, given c_n. The decoding algorithm uses information that the checks provide about the bits, as indicated by the following.

Theorem 1 *For a bit c_n involved in parity checks $\{z_m, m \in \mathcal{M}_n\}$, if the checks are independent then*

$$q_n(x) = \alpha \Pr(c_n = x|y_n) \prod_{m \in \mathcal{M}_n} \Pr(z_m = 0|c_n = x, \mathbf{y}), \tag{13}$$

where α is a normalizing constant.

Proof

$$q_n(x) = \Pr(c_n = 0|\mathbf{y}, \{z_m = 0, m \in \mathcal{M}_n\}) = \frac{\Pr(c_n = 0, \{z_m = 0, m \in \mathcal{M}_n\}|\mathbf{y})}{\Pr(\{z_m = 0\}, m \in \mathcal{M}_n\}|\mathbf{y})}$$
$$= \frac{1}{\Pr(\{z_m = 0\}, m \in \mathcal{M}_n\}|\mathbf{y})} \Pr(c_n = x|\mathbf{y}) \Pr(\{z_m = 0, m \in \mathcal{M}_n\}|c_n = x, \mathbf{y})$$

Due to independence of bits and noise, the conditional probability $\Pr(c_n = x|\mathbf{y})$ can be written as $\Pr(c_n = x|y_n)$. Under the assumption that the checks are independent, the joint probability on the checks can be factored, so that

$$q_n(x) = \frac{1}{\Pr(\{z_m = 0\}, m \in \mathcal{M}_n\})} \Pr(c_n = x|y_n) \prod_{m \in \mathcal{M}_n} \Pr(z_m = 0|c_n = x, \mathbf{y}).$$

The factor dividing this probability can be obtained by marginalizing:

$$q_n(x) = \frac{\Pr(c_n = x|y_n) \prod_{m \in \mathcal{M}_n} \Pr(z_m = 0|c_n = x, \mathbf{y})}{\sum_x \Pr(c_n = x|y_n) \prod_{m \in \mathcal{M}_n} \Pr(z_m = 0|c_n = x, \mathbf{y})};$$

that is, the factor is just a normalization constant, which we denote as α, which ensures that $q_n(x)$ is a probability mass function with respect to x. □

In (13), $q_n(x)$ has two probability factors. The factor $\prod_{m \in \mathcal{M}_n} \Pr(z_m = 0|c_n = x, \mathbf{y})$ has been called the *extrinsic probability* [26]. Like the extrinsic probability used in turbo code decoding [30], it expresses the amount of information there is about c_n based on the structure imposed by the code. The other factor in (13), $P(c_n|y_n)$, expresses how much information there is about c_n based on the measured channel output y_n corresponding to c_n.

As before, let

$$r_{mn}(x) = \Pr(z_m = 0|c_n = x, \mathbf{y}) \tag{14}$$

denote the probability that the mth check is satisfied, given bit c_n. We will derive in section an expression to compute this probability. Using (14) and theorem 1 we can write

$$q_n(x) = \alpha \Pr(c_n = x|\mathbf{y}) \prod_{m \in \mathcal{M}_n} r_{mn}(x). \tag{15}$$

If the Tanner graph were actually a tree with a single tier, as in figure 3(a), theorem 1 could be used immediately for decoding. However, each bit in tier 1 of the tree has its own set of checks, each with their own corresponding checked bits. This leads to a situation as in figure 4. To do decoding on this tree, we again invoke an independence assumption: the set of bits connected to a check subtree rooted at a bit in the first tier are independent. (As before, cycles in the Tanner graph violate this assumption.) The probability of a bit in the first tier of the tree is computed using using bits from the second tier of the tree. Let n' be the index of a bit in the first tier connected to the check z_m. Let

$$q_{mn'}(x) = \Pr(c_{n'} = x | \text{all checks involving } c_{n'}, \text{ except for check } z_m)$$

or, more briefly,

$$q_{mn'}(x) = \Pr(c_{n'} = x | \{z_{m'}, m' \in \mathcal{M}_{n',m}\}, \mathbf{y}).$$

Then, slightly modifying the results of theorem 1,

$$q_{mn'}(x) = \alpha \Pr(c_{n'} = x | y_{n'}) \prod_{m' \in \mathcal{M}_{n',m}} r_{m'n'}(x). \tag{16}$$

If there are w_c parity checks associated with each bit, then the computation (16) involves $w_c - 1$ checks. Using (16), the probabilities for bits in the first tier can be computed from the checks in the second tier, following which the probability at the root c_n can be computed using (15).

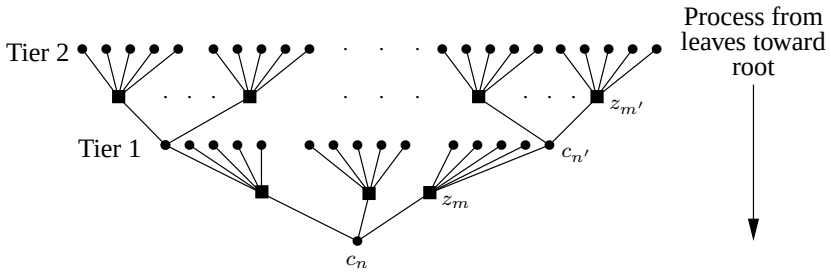


Figure 4: The two-tier tree

Since the product in (16) is computed down the columns of the A matrix (across the checks), updating $q_{mn}(x)$ is called the *vertical step* of the decoding algorithm. Equation (16) computed for each nonzero element of A corresponds to (12). The process can be described in words as follows: For each nonzero position (m, n) of A, compute the product of $r_{m',n}(x)$ down the nth column of A, excluding the value at position (m, n), then multiply by the channel posterior probability. There are $w_c N$ values of q_{mn} to update, each requiring $O(w_c)$ operations, so this step has $O(N)$ complexity.

If the graph associated with the code were actually a tree with independence among the bits associated with the checks on each tier, this procedure could be recursively applied, starting at the leaf nodes of the tree — those not connected to further checks — and working toward the root. The probabilities at the leaf nodes could be computed using the channel posterior probabilities,

$p_n(x)$ defined in (7). Working from leaves toward the root, the probability $q_{mn'}(x)$ would be computed for each node $c_{n'}$ on the second-to-last tier, using the leaf nodes (the last tier). Then $q_{mn'}$ would be computed for each node on the third-to-last tier, using the probabilities obtained from the second-to-last tier, and so forth until the root node is reached.

However, it is time to face reality: The graph associated with the code is not actually a tree. The node c_n which we have called the root of the tree is not a distinguished root node, but is actually an arbitrary node. We deal with this reality by considering each node c_n in turn as if it were the "root" of a tree. For each c_n, we consider each parity check z_m, $m \in \mathcal{M}_n$ associated with it, and compute $q_{mn'}(x)$, as defined in (16), involving other $w_c - 1$ checks and the other bits of the first tier of the "tree" with that c_n as the root. The algorithm does not actually propagate information from leaf to root, but instead propagates information throughout the graph as if each node were the root. If there were no cycles in the tree, this algorithm would, in fact, result in an exact computation at each node of the tree [31]. But as bits connect to checks to other bits through the iterations, there must eventually be some cycles in the graph. These violate the independence assumptions and lead to only approximate results. Nevertheless, it has been found [5] that the approximation is very good, and excellent decoded results are obtained by this approximate algorithm.

Horizontal Step: Updating $r_{mn}(x)$

The probability of the satisfaction of the check z_m, $r_{mn}(x) = \Pr(z_m = 0 | c_n = x, \mathbf{y})$, depends on all of the bits $\{c_{n'}, n' \in \mathcal{N}_m\}$ that participate in z_m, so an expression involving all of the bits that influence check z_m is necessary. The desired probability can be computed by marginalizing,

$$\Pr(z_m = 0 | c_n = x, \mathbf{y}) = \sum_{\{x_{n'}, n' \in \mathcal{N}_{m,n}\}} \Pr(z_m = 0, \{c_{n'} = x_{n'}, n' \in \mathcal{N}_{m,n}\} | c_n = x, \mathbf{y}), \qquad (17)$$

where the sum is taken over all possible binary sequences $\{x_{n'}\}$ with $n' \in \mathcal{N}_{m,n}$. (Continuing example 1, for $\mathcal{N}_{2,1} = \{3, 5, 6, 8, 9\}$, the variables in the sum $\{x_3, x_5, x_6, x_8, x_9\}$ take on all 2^5 possible binary values, from $(0, 0, 0, 0, 0)$ through $(1, 1, 1, 1, 1)$.) The joint probability in (17) can be factored using conditioning as

$$r_{mn}(x) = \sum_{\{x_{n'} : n' \in \mathcal{N}_{m,n}\}} \Pr(z_m = 0 | c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}, \mathbf{y}) \Pr(\{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\} | \mathbf{y}). \qquad (18)$$

Under the assumption that the bits $\{c_{n'}, n' \in \mathcal{N}_{m,n}\}$ are independent — which strictly speaking is true only if there are no cycles in the graph — (18) can be written as

$$r_{mn}(x) = \sum_{\{x_{n'} : n' \in \mathcal{N}_{m,n}\}} \Pr(z_m = 0 | c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}) \prod_{l' \in \mathcal{N}_{m,n}} \Pr(c_{l'} = x_{l'} | \mathbf{y}), \qquad (19)$$

where the conditioning on $\mathbf{y}$ in the first probability has been dropped since the parity is independent of the observations, given the values of the bits. The conditional probability in the sum, $\Pr(z_m = 0 | c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\})$ is either 0 or 1, depending on whether the check condition for

the bits in $\mathcal{N}_m$ is actually satisfied. Only those terms for which $\sum_{n' \in \mathcal{N}_m} x_{n'} = 0$ — or, in other words, for which $x_n = \sum_{n' \in \mathcal{N}_{m,n}} x_{n'}$ — contribute to the probability, so

$$r_{mn}(x) = \sum_{\{\{x_{n'}, n' \in \mathcal{N}_{m,n}\} : x = \sum_l x_l\}} \prod_{l \in \mathcal{N}_{m,n}} \Pr(c_l = x_l | \mathbf{y}). \tag{20}$$

That is, $r_{mn}(x)$ is the total probability of sequences $\{x_{n'}, n' \in \mathcal{N}_{m,n}\}$ of length $|\mathcal{N}_{m,n}|$ whose sum is equal to x, where each element $x_{n'}$ in each sequence occurs with probability $\Pr(c_{n'}|\mathbf{y})$. At first appearance, (20) seems like a complicated sum to compute. However, there are significant computational simplifications that can be made using a graph associated with this problem.

For a sequence of bit values $\{x_{n'}, n' \in \mathcal{N}_{m,n}\}$ involved in check m, let

$$\zeta(k) = \sum_{i=1}^{k} x_{\mathcal{N}_{m,n}(i)}$$

be the sum of the first k bits of $\{x_{n'}\}$. Then

$$\zeta(L) = \sum_{i=1}^{L} x_{\mathcal{N}_{m,n}(i)} = \sum_{l \in \mathcal{N}_{m,n}} x_l, \tag{21}$$

where $L = |\mathcal{N}_{m,n}|$. The sum in (21) appears in the index of the sum in (20).

The values of $\zeta(k)$ as a function of k may be represented using the graph[5] shown in figure 5. There are two states in the trellis, 0 and 1, representing the possible values of $\zeta(k)$. Starting from the 0 state at $k = 0$, $\zeta(1)$ takes on two possible values, depending on the value of $x_{\mathcal{N}_{m,n}(1)}$. Then $\zeta(2)$ takes on two possible values, depending on the value of $\zeta(1)$ and $x_{\mathcal{N}_{m,n}(2)}$. The final state of the trellis represents $\zeta(L)$.

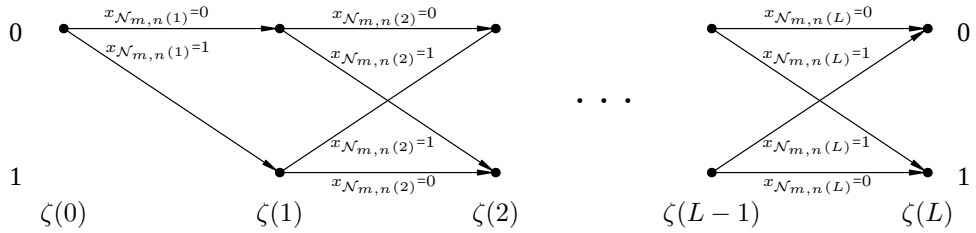


Figure 5: The trellis associated with finding $r_{mn}(x)$. The probability of all possible paths through the trellis is computed inductively.

Let the values $x_{n'}$ occur with probability $\Pr(x_{n'}) = \Pr(c_{n'}|\mathbf{y})$. Using this graph, it may be observed that $r_{mn}(x)$ is the probability that $x = \zeta(L)$, where the probability is computed over *every possible*

[5]Because of its regular structure, a graph of this form is called a trellis.

path through the trellis. The problem still looks quite complicated. However, due to the structure of the trellis, the probabilities can be recursively computed. Note that

$$\Pr(\zeta(1) = x) = \Pr(x_{\mathcal{N}_{m,n}(1)} = x) = \Pr(c_{\mathcal{N}_{m,n}(1)} = x|\mathbf{y}),$$

Knowing $\Pr(\zeta(1) = 0)$ and $\Pr(\zeta(1) = 1)$, the event that $\zeta(2) = 0$ can occur in two ways: either $\zeta(1) = 0$ and $x_{\mathcal{N}_{m,n}(2)} = 0$, or $\zeta(1) = 1$ and $x_{\mathcal{N}_{m,n}(2)} = 1$. Similarly, the event that $\zeta(2) = 1$ can occur in two ways. Thus,

$$\begin{aligned}\Pr(\zeta(2) = 0) &= \Pr(\zeta(1) = 0, x_{\mathcal{N}_{m,n}(2)} = 0) + \Pr(\zeta(1) = 1, x_{\mathcal{N}_{m,n}}(2) = 1)\\ \Pr(\zeta(2) = 1) &= \Pr(\zeta(1) = 1, x_{\mathcal{N}_{m,n}(2)} = 0) + \Pr(\zeta(1) = 0, x_{\mathcal{N}_{m,n}}(2) = 1).\end{aligned} \tag{22}$$

By the (assumed) independence of the bits, the joint probabilities in (22) can be factored, so (22) can be written

$$\begin{aligned}\Pr(\zeta(2) = 0) &= \Pr(\zeta(1) = 0)\Pr(x_{\mathcal{N}_{m,n}(2)} = 0) + \Pr(\zeta(1) = 1)\Pr(x_{\mathcal{N}_{m,n}}(2) = 1)\\ \Pr(\zeta(2) = 1) &= \Pr(\zeta(1) = 1)\Pr(x_{\mathcal{N}_{m,n}(2)} = 0) + \Pr(\zeta(1) = 0)\Pr(x_{\mathcal{N}_{m,n}}(2) = 1).\end{aligned} \tag{23}$$

Let $w_k(x)$ be the probability

$$w_k(x) = \Pr(\zeta(k) = x).$$

Then (23) can be written

$$\begin{aligned}w_2(0) &= w_1(0)\Pr(x_{\mathcal{N}_{m,n}(2)} = 0) + w_1(1)\Pr(x_{\mathcal{N}_{m,n}(2)} = 1)\\ w_2(1) &= w_1(1)\Pr(x_{\mathcal{N}_{m,n}(2)} = 0) + w_1(0)\Pr(x_{\mathcal{N}_{m,n}(2)} = 1),\end{aligned}$$

and, in general (under assumptions of independence)

$$\begin{aligned}w_k(0) &= w_{k-1}(0)\Pr(x_{\mathcal{N}_{m,n}(k)} = 0) + w_{k-1}(1)\Pr(x_{\mathcal{N}_{m,n}(k)} = 1)\\ w_k(1) &= w_{k-1}(1)\Pr(x_{\mathcal{N}_{m,n}(k)} = 0) + w_{k-1}(0)\Pr(x_{\mathcal{N}_{m,n}(k)} = 1).\end{aligned} \tag{24}$$

This recursion is initialized with $w_0(0) = 1$, $w_0(1) = 0$. (It may be noted that the recursion (26) is a particular instance of the celebrated BCJR algorithm, named after its originators [32], which also appears in turbo decoding and hidden Markov model scoring, and a variety of other applications. In this case, the w probabilities are directly analogous to the forward probabilities of the BCJR algorithm, usually denoted by α.)

By the recursive computation (24), the probabilities of all possible paths through the trellis are computed. By the definition of $w_k(x)$, $\Pr(\zeta(L) = x) = w_L(x)$ and by (20),

$$r_{mn}(0) = w_L(0) \qquad r_{mn}(1) = w_L(1). \tag{25}$$

Now consider computing $\Pr(z_m = 0|c_n = x, \mathbf{y})$ for a check node z_m in the first tier of a multi-tier tree, such as that portrayed in figure 4. In this case, the bit probabilities $\Pr(x'_n) = \Pr(c_{n'}|y_{n'})$ necessary for the recursion (24) are obtained from the bit nodes in Tier 1 of the tree, which depend also upon check nodes in Tier 2, excluding check z_m. That is, we use

$$\Pr(c_{n'} = x|\mathbf{y}) = q_{mn'}(x)$$

as the bit probabilities. Then the recursive update rule is written

$$\begin{aligned} w_k(0) &= w_{k-1}(0)q_{m,\mathcal{N}_{m,n}(k)}(0) + w_{k-1}(1)q_{m,\mathcal{N}_{m,n}(k)}(1) \\ w_k(1) &= w_{k-1}(1)q_{m,\mathcal{N}_{m,n}(k)}(0) + w_{k-1}(0)q_{m,\mathcal{N}_{m,n}(k)}(1). \end{aligned} \tag{26}$$

This recursive computation is used for each iteration of the decoding algorithm.

The probability update algorithm (26) can be extended to codes with larger than binary alphabets, by creating a trellis with more states. For binary codes, however, (26) can be re-expressed in an elegant way in terms of differences of probabilities. Let $\delta q_{ml} = q_{ml}(0) - q_{ml}(1)$ and $\delta r_{ml} = r_{ml}(0) - r_{ml}(1)$. From (26),

$$w_k(0) - w_k(1) = (w_{k-1}(0) - w_{k-1}(1))(q_{m\mathcal{N}_{m,n}(k)}(0) - q_{m\mathcal{N}_{m,n}(k)}(1)).$$

Inductively,

$$w_k(0) - w_k(1) = \prod_{i=1}^{k} (q_{m\mathcal{N}_{m,n}(i)}(0) - q_{m\mathcal{N}_{m,n}(i)}(1)).$$

Using $\delta r_{mn} = r_{mn}(0) - r_{mn}(1)$, we have

$$\delta r_{mn} = \prod_{l' \in \mathcal{N}_{m,n}} \delta q_{m,l'}. \tag{27}$$

This is (11). In words, what this update says is: For each nonzero element (m, n) of A, compute the product of the $\delta q_{mn'}$ across the mth row, except for the value at column n. The entire update has complexity $O(N)$.

Having found the δr_{mn} and using the fact that $r_{mn}(0) + r_{mn}(1) = 1$, the probabilities can be computed as

$$r_{mn}(0) = (1 + \delta r_{mn})/2 \qquad r_{mn}(1) = (1 - \delta r_{mn})/2. \tag{28}$$

Terminating and initializing the decoding algorithm

As before, let $q_n(x) = \Pr(c_n = x | \{z_m : m \in \mathcal{M}_n\})$. This can be computed as

$$q_n(x) = \alpha_n p_n(x) \prod_{m \in \mathcal{M}_n} r_{mn}(x),$$

where α_n is chosen so that $q_n(0) + q_n(1) = 1$. These pseudoposterior probabilities are used to make decisions on x: if $q_n(1) > 0.5$ a decision is made to set $\hat{c}_n = 1$.

Since the decoding criterion computes $\Pr(c_n = x | \mathbf{y}, \text{checks involving } c_n)$, with each bit probability computed separately, it is possible that the set of bit decisions obtained by the decoding algorithm will not initially *simultaneously* satisfy all of the checks. This observation can be used

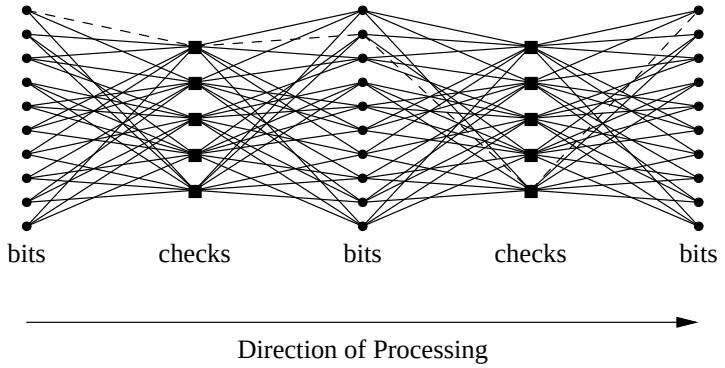


Figure 6: Processing information through the graph determined by A. The dashed line illustrates a cycle of length 4.

to formulate a stopping criterion. If $A\hat{\mathbf{c}} = 0$, that is, all checks are simultaneously satisfied, then decoding is finished. Otherwise, the algorithm repeats from the horizontal step.

Having the ability to determine when decoding is complete distinguishes LDPC codes from other iteratively-decoded codes such as turbo codes, in which either a fixed number of iterations is used, or convergence is declared when the probabilities computed in the algorithm cease changing significantly from one iteration to the next [9, 10, 11].

It may happen that $A\hat{\mathbf{c}} = 0$ is not satisfied after the specified maximum number of iterations. In this case, a *decoding failure* is declared; this is indicative of an error event which exceeds the ability of the code to correct within that number of iterations. Knowledge of a decoding failure is important, but not available with many codes, including turbo codes. In some systems, a decoding failure may invoke a retransmission of the faulty codeword.

The iterative decoding algorithm is initialized by setting $q_{mn}(x) = p_n(x)$. That is, the probability conditional on the checks $q_{mn}(x)$ is set to the channel posterior probability, which is what would be used if the Tanner graph were actually a tree.

Message passing viewpoint

The decoding algorithm can be viewed as an instance of a message passing algorithm, of the sort used for inference in the artificial intelligence literature [11, 22, 23, 31, 33]. In this viewpoint, messages are passed among the nodes in the Tanner graph. In the horizontal step, "messages" in the form of probability vectors $q_{mn}(x)$ are passed to the check nodes, where the messages are combined using (27). In the vertical step, "messages" in the form of probability vectors $r_{mn}(x)$ are passed to the bit nodes, where the messages are combined using (16). The iteration of the algorithm may be viewed as message passing through the graph obtained by concatenating several copies of the Tanner graph, as shown in figure 6.

In the absence of cycles, such message passing algorithms compute exact probabilities [33]. However, the presence of cycles in this graph means that the decoding algorithm computes only approximate solutions. Careful analysis of graphs with cycles, however, [34] has shown theoretically that the approximate algorithm still provides effective decoding capability; this conclusion is borne out by repeated simulation studies.

A numerical example

For the parity check matrix (3) of example 1 and the received probability vector of example 4, the decoding proceeds as follows:

Initialization: Set $q_{mn}(x) = p_n(x)$ from (9)

$$q_{mn}(1)$$
$$\begin{bmatrix} 0.22 & 0.16 & 0.19 & & & 0.87 & 0.18 & & & 0.76 \\ 0.22 & & 0.19 & & 0.55 & 0.87 & & 0.79 & 0.25 & \\ & & 0.19 & 0.48 & 0.55 & & 0.18 & & 0.25 & 0.76 \\ & 0.16 & & 0.48 & 0.55 & 0.87 & & 0.79 & & 0.76 \\ 0.22 & 0.16 & & 0.48 & & & 0.18 & 0.79 & 0.25 & \end{bmatrix}$$

Iteration 1: Horizontal Step:

$$\delta r_{mn}$$
$$\begin{bmatrix} 0.1 & 0.086 & 0.094 & & & -0.079 & 0.091 & & & -0.11 \\ -0.013 & & -0.012 & & 0.075 & 0.01 & & 0.013 & -0.015 & \\ & & 0.00067 & 0.01 & -0.0041 & & 0.00064 & & 0.00083 & -0.00079 \\ & 0.00089 & & 0.015 & -0.0061 & -0.00082 & & -0.001 & & -0.0012 \\ -0.005 & -0.0042 & & -0.071 & & & -0.0044 & 0.0049 & -0.0057 & \end{bmatrix}$$

$$r_{mn}(1)$$
$$\begin{bmatrix} 0.45 & 0.46 & 0.45 & & & 0.54 & 0.45 & & & 0.56 \\ 0.51 & & 0.51 & & 0.46 & 0.49 & & 0.49 & 0.51 & \\ & & 0.5 & 0.49 & 0.5 & & 0.5 & & 0.5 & 0.5 \\ & 0.5 & & 0.49 & 0.5 & 0.5 & & 0.5 & & 0.5 \\ 0.5 & 0.5 & & 0.54 & & & 0.5 & 0.5 & 0.5 & \end{bmatrix}$$

Iteration 1: Vertical Step:

$$q_{mn}(1)$$
$$\begin{bmatrix} 0.23 & 0.16 & 0.19 & & & 0.87 & 0.18 & & & 0.76 \\ 0.19 & & 0.16 & & 0.56 & 0.89 & & 0.79 & 0.25 & \\ & & 0.17 & 0.51 & 0.52 & & 0.16 & & 0.26 & 0.8 \\ & 0.14 & & 0.51 & 0.51 & 0.88 & & 0.78 & & 0.8 \\ 0.19 & 0.14 & & 0.47 & & & 0.15 & 0.79 & 0.26 & \end{bmatrix}$$

$$q_n(1)$$
$$\begin{bmatrix} 0.19 & 0.14 & 0.17 & 0.5 & 0.52 & 0.88 & 0.16 & 0.78 & 0.26 & 0.8 \end{bmatrix}$$

$$\hat{\mathbf{c}} \qquad\qquad \mathbf{z}$$
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

At the end of the first iteration, the parity check condition is not satisfied. The algorithm runs through two more iterations (not shown here). At the end, the decoded value

$$\hat{\mathbf{c}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

is obtained, which exactly matches the transmitted code vector $\mathbf{c}$ of (8).

Even though the minimum distance of the code is 4, the code was able to decode beyond the minimum distance (in this case), and correct two errors.

More on code construction

The code construction described previously is very simple: simply generate columns at random with the appropriate weight. However, there are some practicalities to be dealt with. First, if the columns of A are not linearly independent, some columns can be eliminated, which serves to *increase* the rate of the code by decreasing N. Second, it is important to reduce the number of cycles in the graph associated with the code. Therefore, eliminating or regenerating columns which would contribute to short cycles is advised. It can be seen that when two columns of A have an overlap of more than 1 bit (as in the italicized elements of (3)) there is a cycle of length 4 in the iterated graph. For large N, this is a very easy condition to check for and eliminate in the random generation process. With somewhat more work, longer cycles can be detected and removed.

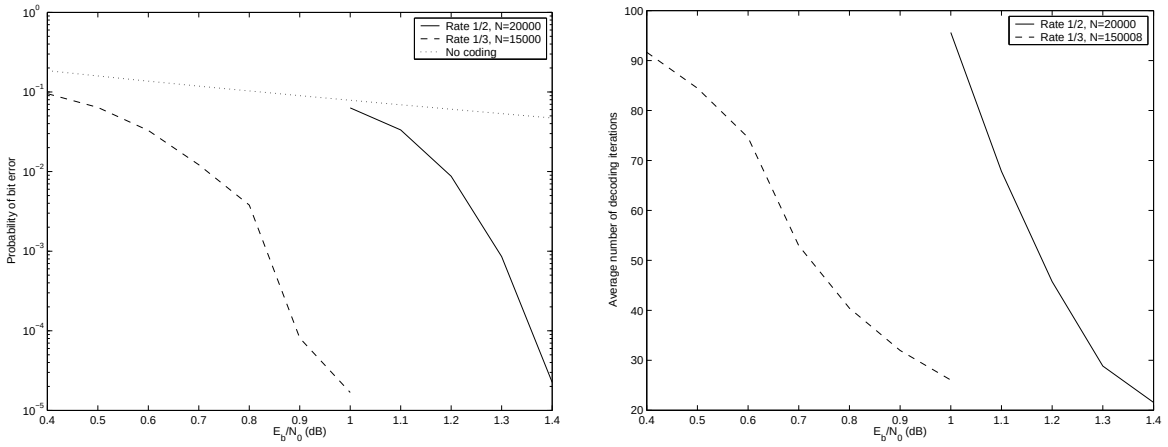
Why low-density parity-check codes?

The iterative decoding algorithm described in this paper applies to any parity check code. For example, Hamming codes are able to benefit from decoding using this algorithm; [26] reports coding gains of 1.5 dB using the iterative decoding algorithm compared with conventional hard decoding of the (7,4) Hamming decoder. Since any linear code can be decoded with this algorithm, why, then, should low-density parity-check codes be contemplated?

First, the codes have excellent distance properties. Gallager showed that the minimum distance $d_{\min}$ between codewords increases with N when column and row weights are held fixed [3, p. 5], that is, as they become increasingly sparse. Sequences of LDPC codes as $N \to \infty$ have been proven to reach channel capacity [5]. (Incidentally, these proofs establish the fact that there is no "error floor" for these codes.) LDPC codes thus essentially act like the random codes used in Shannon's original proof of the channel coding theorem.

Second, the decoding algorithm is tractable. As observed, the decoding algorithm has complexity linearly proportional to the length of the code. (For a nonsparse parity check matrix, the decoding complexity would be proportional to N^2.) Thus we get the benefit of a random code, but without the exponential decoding complexity usually associated with random codes. These codes thus fly in the face of conventional coding wisdom, that there are "few known constructive codes that are good, fewer still that are practical, and none at all that are both practical and very good." [5, p. 399]. While the iterative decoder algorithm may be applied to any parity check code, it is the extreme sparseness of the parity check matrix for LDPC codes that makes the decoding particularly attractive. The low-density nature of the parity check matrix thus, fortuitously, contributes both to good distance properties and the relatively low complexity of the decoding algorithm.

For finite length (but still long codes), excellent coding gains are achievable as we briefly illustrate with two examples. Figure 7(a) shows the BPSK probability of error performance for two LDPC codes, a rate 1/2 code with $(N, K) = (20000, 10000)$ and a rate 1/3 code with $(N, K) = (15000, 10000)$ from [13], compared with uncoded BPSK. These plots were made by adding simulated Gaussian noise to a codeword then iterating the algorithm up to 1000 times. As many as 100000 blocks of bits were simulated to get the performance points at the higher SNRs. In all cases, the errors counted in the probability of error are *detected* errors; in no case did the decoder declare a successful decoding that was erroneous! (This is not always the case. We have found that for very short toy codes, the decoder may terminate with the condition $A\hat{\mathbf{c}} = \mathbf{0}$, but $\hat{\mathbf{c}}$ is erroneous. However, for long codes, decoding success meant correct decoding.)



(a) Performance for a rate 1/2 and a rate 1/3 code

(b) Average number of decoding iterations

Figure 7: Illustration of the decoding performance of LPDC codes, and the number of iterations to achieve decoding.

Figure 7(b) shows the *average* number of iterations to complete the decoding. (The peak number of iterations, not shown, was in many instances much higher.) The high number of iterations suggests a rather high potential decoding complexity, even though each iteration is readily computed. It has been estimated [5, p. 420] that the decoding complexity per bit is $6w_c/R$ multiplies per bit per iteration. For the $R = 1/2$ code with $w_c = 3$, this is 36 multiplies per bit per iteration. For 50 iterations, this is 1800 multiplies per decoded bit. For comparison, the turbo code of [2] is reported to require 3800 operations per decoded bit.

There are some potential disadvantages to LDPC codes. First, the best code performance is obtained for very long codes (as predicted by the channel coding theorem). This long block length, combined with the need for iterative decoding, introduces latency which is unacceptable in many applications. Second, the encoding operation is quite involved, since the G matrix is not necessarily sparse, so the encoding complexity is $O(N^2)$. Work is underway to find structured encoder matrices that can encode with lower complexity or to otherwise efficiently encode; see e.g., [35, 36].

It should also be pointed out that, while this introduction has focused on regular binary codes, the best known LDPC codes are irregular and over $GF(2^m)$ for $m = 3$ or 4.

Summary and Conclusion

LDPC codes are very strong codes, capable of approaching channel capacity. It is easy to create a parity check matrix for them, and the decoding algorithm has complexity only linear in the code length. It is expected that LDPC codes will be used in a variety of upcoming applications. Work is underway to implement LDPC in DSL, wireless links, and high density magentic recording applications, as well as applications to space-time coding.

*

References

[1] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical J.*, vol. 27, pp. 379–423, 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. of the 1993 International Conference on Communications*, pp. 1064–1070, 1993.

[3] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan 1962.

[4] R. G. Gallager, *Low-density parity-check codes*. Cambridge, MA: M.I.T. Press, 1963.

[5] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Info. Theory.*, vol. 45, pp. 399–431, March 1999.

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

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

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

[9] D. Agrawal and A. Vardy, "The turbo decoding algorithm and its phase trajectories," *IEEE Trans. Information Theory*, 2001.

[10] R. Shao, M. Fossorier, and S. Lin, "Two simple stopping criteria for iterative decoding," in *Proc. IEEE Symp. Info. Theory*, (Cambridge, MA), p. 279, Aug. 1998.

[11] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Info. Theory.*, vol. 42, pp. 429–445, Mar. 1996.

[12] C. Schlegel, *Trellis Coding*. New York: IEEE Press, 1997.

[13] D. Mackay. `http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html`.

[14] D. J. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding 5th IMA Conference* (C. Boyd, ed.), vol. 1025 of *Lecture Notes in Computer Science*, pp. 100–111, Springer, Spring 1995.

[15] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Info. Theory.*, vol. 27, pp. 533–547, Sept. 1981.

[16] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Euro. Trans. Telecommun.*, vol. 6, pp. 513–525, 1995.

[17] N. Wiberg, *Codes and decoding on general graphs*. PhD thesis, Linköping University, 1996.

[18] M. C. Davey and D. J. MacKay, "Low density parity check codes over $GF(q)$," *IEEE Com. Letters*, vol. 2, no. 6, 1998.

[19] D. J. MacKay, "Near Shannon limit performance of low density parity check codes," *Eletronics Letters*, vol. 33, pp. 457–458, Mar 1997.

[20] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 47, pp. 638–656, Feb. 2001.

[21] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Trans. Information Theory*, vol. 47, pp. 657–670, Feb. 2001.

[22] G. D. Forney, "Codes on graph: Normal realizations," *IEEE Trans. Info. Theory.*, vol. 47, pp. 520–548, Feb. 2001.

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

[24] P. Sweeney, *Error Control Coding*. New York, NY: Wiley, 2002.

[25] R. H. Zaragoza, *The Art of Error Control Coding*. Wiley, 2002.

[26] J. R. Barry, "Low-density parity-check codes." `http://users.ece.gatech.edu/~barry/6606/handouts/ldpc.pdf`, Oct. 2001.

[27] R. G. Gallager, "A perspective on multiaccess channels," *IEEE Transactions on Information Theory*, vol. IT-31, no. 2, pp. 124–142, March 1985.

[28] M. G. Luby, M. Miztenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Information Theory*, vol. 47, pp. 585–598, Feb. 2001.

[29] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the intractability of certain coding problems," *IEEE Trans. Information Theory*, vol. 24, pp. 384–386, May 1978.

[30] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcing coding and decoding: Turbo codes," in *Proc. 1993 IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 1064–1070, 1993.

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

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

[33] J. Pearl, *Probalistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann, 1988.

[34] Y. Weiss, "Correctness of local porbability propagation in graphical models with loops," *Neural Computation*, vol. 12, pp. 1–41, 2000.

[35] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 47, pp. 638–656, Feb. 2001.

[36] D. Haley, A. Grant, and J. Buetefuer, "Iterative encoding of low-density parity-check codes," in *IEEE Globecom*, (Taipei,Taiwan), Nov. 2002.