Low-Density Parity-Check Codes

John R. Barry
Georgia Institute of Technology
barry@ece.gatech.edu

October 5, 2001

We present a tutorial overview of low-density parity-check (LDPC) codes. As the name suggests, LDPC codes are block codes defined by a parity-check matrix that is sparse. They were first proposed in 1962 by Gallager [1][2], along with an elegant iterative decoding scheme whose complexity grows only linearly with block length. Despite their promise, LDPC codes were largely forgotten for several decades, primarily because the computers at the time were not powerful enough to exploit them. In 1995 they were rediscovered by MacKay and Neal [3], sparking a flurry of further research. Today the value of LDPC codes is widely recognized. Their remarkable performance ensures that they will not be forgotten again. In contrast to many codes that were invented well after 1962, LDPC codes offer both better performance *and* lower decoding complexity. In fact, it is an irregular LDPC code (with block length 10^7) that currently holds the distinction of being the world's best-performing rate-$1/2$ code, outperforming all other known codes, and falling only 0.04 dB short of the Shannon limit [4].

1. Parity-Check Codes

A *parity-check code* of length N is a linear binary block code whose codewords all satisfy a set of M linear parity-check constraints. It is traditionally defined by its $M \times N$ *parity-check matrix* $\mathbf{H}$, whose M rows specify each of the M constraints. For example, if the first constraint specifies that bits 3 and 7 must be equal, then the first row of $\mathbf{H}$ contains a 1 in position 3 and 7 and zeros elsewhere. The parity-check code is the set of binary vectors satisfying all constraints, i.e., the set of $\boldsymbol{c}$ satisfying $\boldsymbol{c}\mathbf{H}^T = \mathbf{0}$. Each linearly independent constraint cuts the number of valid codewords in half. Thus, if $r = \mathrm{rank}(\mathbf{H}) \le M$ is the number of linearly independent rows in $\mathbf{H}$, then the number of codewords is 2^{N-r}, and the code dimension is $K = N - r$. Because each codeword of length N conveys K information bits, the code rate is K/N.

A low-density parity-check (LDPC) code is defined by a parity-check matrix that is sparse [1].

Definition 1. A *regular* (j, k) *LDPC matrix* is an $M \times N$ binary matrix having exactly j ones in each column and exactly k ones in each row, where $j < k$ and both are small compared to N.

(An *irregular* [5] LDPC matrix is still sparse, but not all rows and columns contain the same number of ones. To simplify our discussion we will focus on regular LDPC matrices.)

By this definition, every parity-check equation of a regular LDPC code involves exactly k bits, and every bit is involved in exactly j parity-check equations. The restriction $j < k$ is needed to ensure that more than just the all-zero codeword satisfies all of the constraints, or equivalently, to ensure a nonzero code rate. Indeed, the total number of ones in $\mathbf{H}$ is $Mk = Nj$, since there are M rows, each containing k ones, and there are N columns, each containing j ones. The code rate $R = 1 - M/N$ is then $R = 1 - j/k$, assuming the M rows are linearly independent. The need for $j < k$ is thus clear.

From the equation $Mk = Nj$, we also find that the number of rows in a (j, k) LDPC matrix is $M = Nj/k$. It immediately follows that the parameters N, j, and k cannot be chosen independently, but must be related in such a way that Nj/k is be an integer. For example, a (3, 4) LDPC matrix exists when $N = 1000$ and $N = 1004$, but not when $N = 1002$.

Observe that the fraction of ones in a regular (j, k) LDPC matrix is k/N. The "low density" terminology derives from the fact that this fraction approaches zero as $N \to \infty$. In contrast, the average fraction of ones in a purely random binary matrix (with independent components equally likely to be zero or one) is $1/2$.

An $M \times N$ regular (j, k) LDPC matrix can often (but not always) be conveniently expressed in terms of the following shorter parity-check matrix $\mathbf{H}_0$:

$$\mathbf{H}_0 = \begin{bmatrix} \overbrace{1\ 1\ 1 \ldots 1}^{k} & \overbrace{1\ 1\ 1 \ldots 1}^{k} & \ddots & \overbrace{1\ 1\ 1 \ldots 1}^{k} \end{bmatrix}. \tag{1}$$

It has $N/k = M/j$ rows and N columns. The m-th row contains ones in columns $(m-1)k + 1$ through mk and zeros elsewhere. By itself, $\mathbf{H}_0$ would define a code consisting of N/k *independent* single-parity check constraints, with the first row constraining the parity of the first block of k coded bits, the next row constraining the parity of the second block of k bits, etc. In this code, every parity check involves k bits, and each bit is involved in one and only one parity check. Hence, $\mathbf{H}_0$ alone defines a $(1, k)$ regular LDPC code. However, the performance of this code would be poor. In fact, because the first two columns of $\mathbf{H}_0$ are linearly dependent (or equivalently because 0000… 0 and 1100… 0 are both valid codewords), the minimum distance for the code would be two.

We can construct a regular (j, k) LDPC matrix by stacking j column permutations of $\mathbf{H}_0$ one atop another:

$$\mathbf{H} = \begin{bmatrix} \pi_1(\mathbf{H}_0) \\ \pi_2(\mathbf{H}_0) \\ \\ \pi_j(\mathbf{H}_0) \end{bmatrix}. \tag{2}$$

Here $\pi_i(\mathbf{H}_0)$ denotes a matrix whose columns are a permuted version of the columns of $\mathbf{H}_0$. Since each row of $\mathbf{H}_0$ has k ones, each row of $\mathbf{H}$ also has k ones. Similarly, since each column of $\mathbf{H}_0$ contains a single one, each column of $\mathbf{H}$ contains j ones. We remark that not all (j, k) regular LDPC matrices $\mathbf{H}$ satisfying Defn. 1 can be expressed in the form of (2).

We remark that the building block $\mathbf{H}_0$ in (1) was chosen somewhat arbitrarily; any column permutation of $\mathbf{H}_0$ could have been used in its place. For example, we can equivalently use $\mathbf{H}_0 = [\mathbf{I}\ \mathbf{I}\ \mathbf{I}\ \ldots\ \mathbf{I}]$ in place of (1), where $\mathbf{H}_0$ is a concatenation of k identity matrices $\mathbf{I}_{N/k}$.

A proper choice of the permutations will allow the minimum distance of the code defined by $\mathbf{H}$ to increase beyond two. The prospect of designing j different permutations of length N may at first seem daunting, especially for large N. However, Gallager proved that a totally random choice will on average produce an excellent code [2]. In particular, if each permutation is chosen independently and uniformly from the set of all $N!$ possible permutations, then the expected minimum distance that results will increase linearly with N. A code with such a property is said to be "good." The idea of designing codes randomly did not originate with Gallager, but dates back to Shannon's original work [6]. The beauty of Gallager's design is that, unlike Shannon's random codes, we will see that it is possible to decode Gallager's codes with complexity that grows only linearly with N.

Example 1. The following is an example of a LDPC matrix with wordlength $N = 20$, $j = 3$, and $k = 4$ [1]:

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]. \tag{3}$$

The horizontal lines separate $\mathbf{H}_0$ and its two permutations. We see that each column contains $j = 3$ ones, and each row contains $k = 4$ ones. The corresponding permutations are $\pi_1 = [1\ 2\ \ldots\ 20]$, $\pi_2 = [1\ 5\ 9\ 13\ 2\ 6\ 10\ 17\ 3\ 7\ 14\ 18\ 4\ 11\ 15\ 18\ 8\ 12\ 16\ 20]$, and $\pi_3 = [1\ 5\ 9\ 13\ 17\ 2\ 6\ 10\ 14\ 18\ 7\ 3\ 11\ 15\ 19\ 8\ 16\ 4\ 12\ 20]$. It can be shown that the tenth row of $\mathbf{H}$ is the sum of the first nine rows, and that the fifteenth row is the sum of rows one through five and rows eleven through fourteen. Thus rows ten and fifteen are linearly dependent on the remaining rows. It can be shown that the remaining 13 rows are linearly independent. Hence, the rank of $\mathbf{H}$ is 13, the dimension of the LDPC code is $K = 7$, and the code rate is $K/N = 0.35$.

The previous parity-check matrix has 15 rows, but only 13 of them are independent. We could define a new full-rank parity-check matrix $\tilde{\mathbf{H}}$ by eliminating the redundant rows from $\mathbf{H}$. Then $\tilde{\mathbf{H}}$ would describe the same code as $\mathbf{H}$, since $\boldsymbol{c}\mathbf{H}^T = \mathbf{0}$ if and only if $\boldsymbol{c}\tilde{\mathbf{H}}^T = \mathbf{0}$. However, the number of ones in k columns of $\tilde{\mathbf{H}}$ would decrease each time a redundant row is removed, so that $\tilde{\mathbf{H}}$ would no longer obey the regularity property of a regular LDPC matrix. We will often choose to describe an LDPC code by a rank-deficient but regular LDPC matrix, as opposed to its more efficient but irregular full-rank equivalent.

Any parity-check code (including an LDPC code) may be specified by a *Tanner graph* [7][8], which is essentially a visual representation of the parity check matrix $\mathbf{H}$. Recall that an $M \times N$ parity-check matrix $\mathbf{H}$ defines a code in which the N bits of each codeword satisfy a set of M parity-check constraints. The Tanner graph contains N "bit" nodes, one for each bit, and M "check" nodes, one for each of the parity checks. The bit nodes are depicted using circles, while the check nodes are depicted using squares. The check nodes are connected to the bit nodes they check. Specifically, a branch connects check node m to bit node n if and only if the m-th parity check involves the n-th bit, or more succinctly, if and only if $H_{m,n} = 1$. The graph is said to be *bipartite* because there are two distinct types of nodes, bit nodes and check nodes, and there can be no direct connection between any two nodes of the same type.

Example 2. The Tanner graph associated with the 15×20 LDPC matrix of (3) is shown in Fig. 1. The bit nodes are represented by the $N = 20$ circles at the top, while the check nodes are represented by the $M = 15$ squares at the bottom. The first (left-most) five check nodes correspond to $\mathbf{H}_0$, the second five to $\pi_2(\mathbf{H}_0)$, and the last five to $\pi_3(\mathbf{H}_0)$.

For the special case of a (j, k) regular LDPC code, each bit is involved in j parity checks. Hence, the number of branches emanating from a bit node is always j. Similarly, because each parity check involved k bits, the number of branches emanating from each check node is always k. Observe that the graph of Fig. 1 satisfies these properties.

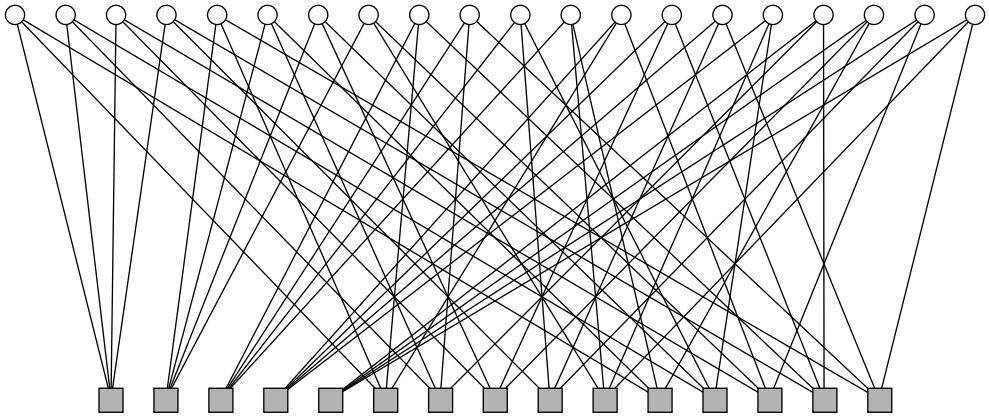

Fig. 1. The Tanner graph for the LPDC matrix of (3).

The value of the Tanner graph will become clear in the next section, where we describe a decoding algorithm for LDPC codes. There, the graph will be used to describe a parallel implementation of a decoder, with the different nodes representing separate processors, and edges representing communication between processors.

2. Decoding Parity-Check Codes

2.1. Background and Terminology

The probability distribution for a binary random variable $c \in \{0, 1\}$ is uniquely specified by the single parameter $p = \Pr[c = 1]$, since $\Pr[c = 0] = 1 - p$. Alternatively, the probability distribution is also uniquely specified by the logarithm of the ratio:

$$\lambda = \log \frac{\Pr[c = 1]}{\Pr[c = 0]}. \tag{4}$$

To recover p from λ, we observe that $e^{\lambda} = p/(1 - p)$, and solving for p yields $p = 1/(1 + e^{-\lambda})$. The sign of λ indicates the most likely value for c; λ is positive when 1 is more likely than 0, and λ is negative when 0 is more likely than 1. Moreover, the magnitude $|\lambda|$ is a measure of certainty. At one extreme, if $\lambda = 0$ then 0 and 1 are equally likely. At the other extreme, if $\lambda = \infty$ then $c = 1$ with probability 1, and $\lambda = -\infty$ implies $c = 0$.

Given a random bit $c \in \{0, 1\}$, let r denote an observation whose pdf depends on c according to $f(r \mid c)$. When c is fixed and $f(r \mid c)$ is viewed as a function of r, it is called a *conditional pdf*. On the other hand, when r is fixed, then $f(r \mid c)$ as a function of c is called the *likelihood function*.

Before making an observation, the *a priori* probabilities for c are $\Pr[c = 1]$ and $\Pr[c = 0]$. After making an observation, these probabilities change to the *a posteriori* probabilities (APP) $\Pr[c = 1 \mid r]$ and $\Pr[c = 0 \mid r]$. Because of Bayes rule, the a posteriori probability is proportional to the likelihood function:

$$\Pr[c = 1 \mid r] = f(r \mid c = 1)\Pr[c = 1]/f(r) . \tag{5}$$

Hence, the logarithm of the ratio of a posteriori probabilities can be expressed as:

$$\log \frac{\Pr[c = 1 \mid r]}{\Pr[c = 0 \mid r]} = \log \frac{f(r \mid c = 1)}{f(r \mid c = 0)} + \log \frac{\Pr[c = 1]}{\Pr[c = 0]}. \tag{6}$$

The first term on the right-hand side is called the *log-likelihood ratio (LLR)*. Strictly speaking, the second term on the right-hand side is a log-probability ratio, and the left-hand side is a log-APP ratio. However, with an abuse of notation, the second term on the right-hand side is more commonly called the *a priori LLR*, and the left-hand side is called the *a posteriori LLR*. If c is equally likely to be zero or one, then the *a priori* LLR is zero, and the *a posteriori* LLR is equal to the LLR.

2.2. The Tanh Rule

Let $\phi(\boldsymbol{c}) \in \{0, 1\}$ denote the *parity* of a set $\boldsymbol{c} = [c_1 \ldots c_n]$ of n bits, so that $\phi(\boldsymbol{c}) = 0$ if there are an even number of ones in $\boldsymbol{c}$, and $\phi(\boldsymbol{c}) = 1$ if there are an odd number. If the bits are independent, the *a priori* LLR for the parity obeys the *tanh* rule [9][4].

Exercise 1. (The Tanh Rule.) Let $\boldsymbol{c} = [c_1 \ldots c_n] \in \{0, 1\}^n$ be a vector of n bits that are independent but not equiprobable; let $\lambda_i = \log(\Pr[c_i = 1] / \Pr[c_i = 0])$ denote the a priori LLR for the i-th bit. Show that the a priori LLR $\lambda_{\phi(\boldsymbol{c})} = \log(\Pr[\phi(\boldsymbol{c}) = 1] / \Pr[\phi(\boldsymbol{c}) = 0])$ for the parity satisfies the so-called "tanh rule:"

$$\tanh\left(\frac{-\lambda_{\phi(\boldsymbol{c})}}{2}\right) = \prod_{i=1}^{n} \tanh\left(\frac{-\lambda_i}{2}\right). \tag{7}$$

Solution. See Appendix A.

Solving (7) for $\lambda_{\phi(\boldsymbol{c})}$ yields the following equivalent relationship:

$$\lambda_{\phi(\boldsymbol{c})} = -2\tanh^{-1}\left(\prod_{i=1}^{n} \tanh\left(\frac{-\lambda_i}{2}\right)\right). \tag{8}$$

Alternatively, if we treat the signs and magnitudes of the LLRs separately, then Appendix B shows that (8) can equivalently be expressed as [1]:

$$\lambda_{\phi(\boldsymbol{c})} = -\prod_{i=1}^{n} \operatorname{sign}(-\lambda_i)\, f\left(\sum_{i=1}^{n} f(|\lambda_i|)\right), \tag{9}$$

where we have introduced the special function:

$$f(x) = \log\frac{e^x + 1}{e^x - 1} = -\log(\tanh(x/2)) \ . \tag{10}$$

In hardware implementations, (9) is preferred over (7) or (8), because it involves the sum of n terms instead of the product n terms. Nevertheless, (7) is preferred in analysis because of its conceptual simplicity.

The function $f(x)$ has some interesting properties. As shown in Fig. 2, it is positive and monotonically decreasing for $x > 0$, with $f(0) = \infty$ and $f(\infty) = 0$. Furthermore, $f(x)$ is its own inverse! That is, $f(f(x)) = x$ for all $x > 0$. This property is easily verified by direct substitution.

Let $\hat{\boldsymbol{c}} = [\hat{c}_1 \ldots \hat{c}_n]$ denote the most likely value for $\boldsymbol{c}$, where $\hat{c}_i = 1$ if $\lambda_i > 0$, else $\hat{c}_i = 0$. The sign of $\lambda_{\phi(\boldsymbol{c})}$, which indicates the most likely value for $\phi(\boldsymbol{c})$, is completely determined by $\phi(\hat{\boldsymbol{c}})$, according to:

$$\operatorname{sign}(\lambda_{\phi(\boldsymbol{c})}) \ = \ -\prod_{i=1}^{n} \operatorname{sign}(-\lambda_i) \ = \ -(-1)^{\phi(\hat{\boldsymbol{c}})}. \tag{11}$$

This is to be expected, since the parity is most likely even when an even number of λ_i's are positive, and odd when an odd number are positive.

On the other hand, the magnitude of $\lambda_{\phi(\boldsymbol{c})}$, which measures the certainty that $\phi(\boldsymbol{c})$ is its most likely value, is given by $|\lambda_{\phi(\boldsymbol{c})}| = f(\Sigma_i f(|\lambda_i|))$. Suppose the k-th bit c_k of $\boldsymbol{c}$ is equally likely to be 0 or 1, so that $\lambda_k = 0$. The k-th term in the sum $\Sigma_i f(|\lambda_i|)$ is then infinity, so that the entire sum is infinite. Because $f(\infty) = 0$, it follows that (9) reduces to $\lambda_{\phi(\boldsymbol{c})} = 0$ whenever any one of the bits has zero log-likelihood ratio. This makes intuitive sense, since if one bit is equally likely to be zero or one, then the parity of the entire vector is equally likely to be zero or one, regardless of the probabilities of the remaining bits. More generally, whenever one bit is significantly less certain than the others, so that the summation is dominated by $f(|\lambda_{\min}|)$, where $|\lambda_{\min}| = \min_i\{|\lambda_i|\}$, then the magnitude of $\lambda_{\phi(\boldsymbol{c})}$ simplifies to:

$$\begin{aligned} |\lambda_{\phi(\boldsymbol{c})}| &= f(\Sigma_i f(|\lambda_i|)) \\ &\approx f(f(|\lambda_{\min}|)) \\ &= |\lambda_{\min}|. \end{aligned} \tag{12}$$

The certainty in the parity of a vector can thus be approximated by the certainty of the least certain bit. Substituting (12) into (9) yields the following approximation:

$$\lambda_{\phi(\boldsymbol{c})} \approx (-1)^{\phi(\hat{\boldsymbol{c}})} |\lambda_{\min}|. \tag{13}$$

In the next section we will show how (8) can be used in the decoding problem, but a lower-complexity approximation would use (13) in place of (8).

2.3. The Decoding Problem

Let us consider the problem of decoding a code with parity-check matrix $\mathbf{H}$, given that the channel adds white Gaussian noise, so that the receiver observation $\boldsymbol{r} = [r_1 \ldots r_N]$ is related to the transmitted codeword $\boldsymbol{c} = [c_1 \ldots c_N]$ by:

$$\boldsymbol{r} = 2\boldsymbol{c} - 1 + \boldsymbol{n}\ , \tag{14}$$

where the components of the noise vector $\boldsymbol{n}$ are independent zero-mean Gaussian random variables with variance σ^2. (This implies an antipodal bit-to-symbol mapping $0 \rightarrow -1$, $1 \rightarrow 1$.)

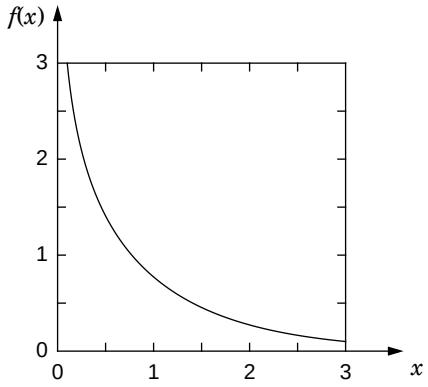

Fig. 2. The function f(x).

The detector that minimizes the probability of error for the n-th bit would calculate the *a posteriori* LLR:

$$\begin{aligned}\lambda_n &= \log\frac{\Pr[c_n = 1 \mid \boldsymbol{r}]}{\Pr[c_n = 0 \mid \boldsymbol{r}]} \\ &= \log\frac{\Pr[c_n = 1 \mid r_n, \{r_{i\neq n}\}]}{\Pr[c_n = 0 \mid r_n, \{r_{i\neq n}\}]},\end{aligned} \tag{15}$$

and then decide $\hat{c}_n = 1$ if $\lambda_n > 0$, and $\hat{c}_n = 0$ otherwise. Applying Bayes rule, the numerator in (15) can be written as:

$$\begin{aligned}\Pr[c_n = 1 \mid r_n, \{r_{i\neq n}\}] &= \frac{f(r_n, c_n = 1, \{r_{i\neq n}\})}{f(r_n, \{r_{i\neq n}\})} \\ &= \frac{f(r_n \mid c_n = 1, \{r_{i\neq n}\}) f(c_n = 1, \{r_{i\neq n}\})}{f(r_n \mid \{r_{i\neq n}\}) f(\{r_{i\neq n}\})} \\ &= \frac{f(r_n \mid c_n = 1)\Pr[c_n = 1 \mid \{r_{i\neq n}\}]}{f(r_n \mid \{r_{i\neq n}\})}.\end{aligned} \tag{16}$$

The last equality exploits the fact that, given c_n, r_n is independent of $\{r_{i\neq n}\}$. The denominator of (15) can be similarly expressed. Hence, (15) simplifies to:

$$\begin{aligned}\lambda_n &= \log\frac{f(r_n \mid c_n = 1)\Pr[c_n = 1 \mid \{r_{i\neq n}\}]}{f(r_n \mid c_n = 0)\Pr[c_n = 0 \mid \{r_{i\neq n}\}]} \\ &= \log\frac{f(r_n \mid c_n = 1)}{f(r_n \mid c_n = 0)} + \log\frac{\Pr[c_n = 1 \mid \{r_{i\neq n}\}]}{\Pr[c_n = 0 \mid \{r_{i\neq n}\}]} \\ &= \underbrace{\frac{2}{\sigma^2} r_n}_{\text{intrinsic}} + \underbrace{\log\frac{\Pr[c_n = 1 \mid \{r_{i\neq n}\}]}{\Pr[c_n = 0 \mid \{r_{i\neq n}\}]}}_{\text{extrinsic}},\end{aligned} \tag{17}$$

where we used the fact that $f(r_n \mid c_n) = (2\pi\sigma^2)^{-1/2}\exp(-(r_n - 2c_n + 1)^2/(2\sigma^2))$ for the AWGN channel.

The first term in (17) represents the contribution from the n-th channel observation, and is called the *intrinsic* information, while the second term represents the contribution from the *other* observations, and is called the *extrinsic* information [10]. Interestingly, the contributions are combined by simply adding. Further, the intrinsic information is proportional to the n-th channel observation. The constant of proportionality $2/\sigma^2$ is called the *channel reliability* [9].

Exercise 2. Consider a binary symmetric channel with input and output alphabet $\{\pm 1\}$. Show that the *a posteriori* LLR is again given by (17), except that the channel reliability is $\log\left(\frac{1-p}{p}\right)$ instead of $2/\sigma^2$, where p is the crossover probability.

Before we can simplify (17) further, we must examine more closely the structure of the Tanner graph. Consider Fig. 3, where we draw the graph of a LDPC code from the perspective of the n-th bit node, and where we have rearranged the bit nodes and check nodes so as to

avoid crossing edges. The n-th bit is involved in exactly j parity checks, numbered 1 through j, and each of the checks involve $k-1$ other bits. As shown in the figure, let $\boldsymbol{c}_{(i)} = [c_{i,2} \ \ldots \ c_{i,k}]$ denote the set of bits involved in the i-th parity check equation, excluding c_n.

A *cycle* is a path through the graph that begins and ends at the same bit node. The length of the cycle is the number of edges traversed. Because the graph is bipartite, the minimum length of a cycle is four. For example, for the graph shown in Fig. 3, the existence of a cycle depends on the dotted edge. With the dotted edge in place, the graph contains a single cycle of length four. However, if we were to remove the dotted edge, the graph would have no cycles. A graph without cycles is a *tree*. A cycle-free graph has the following properties:

- removing any edge creates two separate subgraph.
- there is a unique path connecting any pair of bit nodes.
- As a special case of the above, from the point of view of the bit node c_n: Every bit node is reachable from c_n through one and only one of the edges incident on c_n.
- If bit nodes c_j and c_k are reachable from c_n through different edges, then c_j and c_k are conditionally independent, when the n-th observation is excluded: $\Pr[c_j\,, c_k \mid \{r_{i\neq n}\}] = \Pr[c_j \mid \{r_{i\neq n}\}]\Pr[c_k \mid \{r_{i\neq n}\}]$.

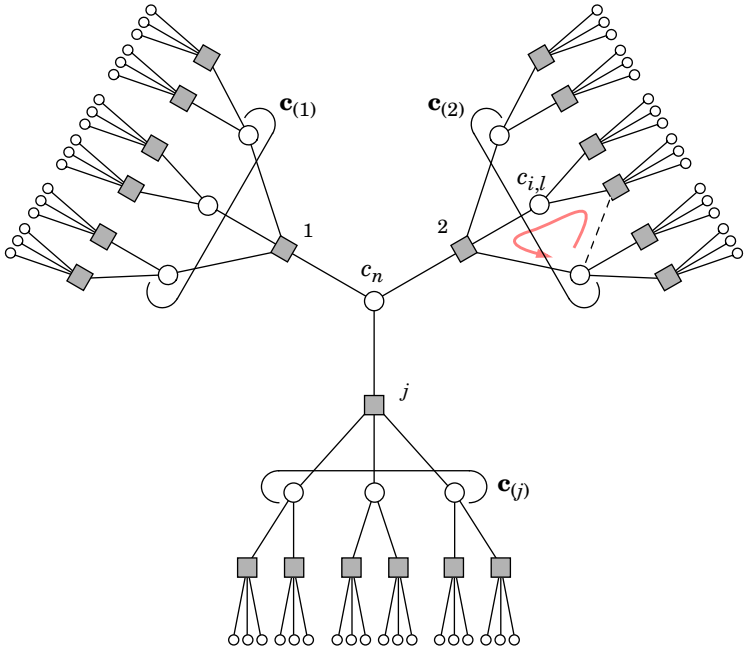

Fig. 3. The graph for an irregular LDPC code, from the perspective of the n-th bit node c_n. Removing any one of the edges incident on c_n would create two separate graphs. In fact, if the dotted edge did not exist, then all edges would have this property, and the graph would form a *tree*. This particular code is not regular, because some bit nodes (for example, the leaf nodes) are involved in only one parity-check equation, while others (for example, the n-th bit node) are involved in more than one.

Because the j parity constraints of the code ensure that $c_n = \phi(\boldsymbol{c}_{(i)})$ for all $i = 1 \ldots j$, we can rewrite (17) as:

$$\lambda_n = \frac{2}{\sigma^2} r_n + \log \frac{\Pr[\phi(\boldsymbol{c}_{(i)}) = 1 \text{ for } i = 1 \ldots j \mid \{r_{i \neq n}\}]}{\Pr[\phi(\boldsymbol{c}_{(i)}) = 0 \text{ for } i = 1 \ldots j \mid \{r_{i \neq n}\}]}. \tag{18}$$

If the graph is cycle-free, the vectors $\boldsymbol{c}_{(1)}$, $\boldsymbol{c}_{(2)}$, …, $\boldsymbol{c}_{(j)}$ are conditionally independent given $\{r_{i \neq n}\}$, and furthermore, the components of $\boldsymbol{c}_{(i)}$ are themselves conditionally independent given $\{r_{i \neq n}\}$. Hence, (18) reduces to:

$$\begin{aligned} \lambda_n &= \frac{2}{\sigma^2} r_n + \log \frac{\prod_{i=1}^{j} \Pr[\phi(\boldsymbol{c}_{(i)}) = 1 \mid \{r_{i \neq n}\}]}{\prod_{i=1}^{j} \Pr[\phi(\boldsymbol{c}_{(i)}) = 0 \mid \{r_{i \neq n}\}]} \\ &= \frac{2}{\sigma^2} r_n + \sum_{i=1}^{j} \log \frac{\Pr[\phi(\boldsymbol{c}_{(i)}) = 1 \mid \{r_{i \neq n}\}]}{\Pr[\phi(\boldsymbol{c}_{(i)}) = 0 \mid \{r_{i \neq n}\}]} \qquad (19) \\ &= \frac{2}{\sigma^2} r_n + \sum_{i=1}^{j} \lambda_{\phi(\boldsymbol{c}_{(i)})}, \qquad (20) \end{aligned}$$

where because the bits are conditionally independent, each $\lambda_{\phi(\boldsymbol{c}_{(i)})}$ satisfies the tanh rule of (8). In particular, if we introduce

$$\lambda_{i,l} = \log \frac{\Pr[c_{i,l} = 1 \mid \{r_{i \neq n}\}]}{\Pr[c_{i,l} = 0 \mid \{r_{i \neq n}\}]}, \tag{21}$$

then substituting (8) into (20) yields:

$$\lambda_n = \frac{2}{\sigma^2} r_n - 2 \sum_{i=1}^{j} \tanh^{-1} \left(\prod_{l=2}^{k} \tanh\left(\frac{-\lambda_{i,l}}{2} \right) \right). \tag{22}$$

With the aid of the Tanner graph of Fig. 3, we may interpret (20) in terms of messages passed from node to node. Suppose the bit node associated with $c_{i,l}$ passes the "message" $\lambda_{i,l}$ to the i-th check node. In turn, the i-th check node collects the $k - 1$ incoming messages from the other bits $\boldsymbol{c}_{(i)}$ involved (beside c_n), computes the *a posteriori* LLR $\lambda_{\phi(\boldsymbol{c}(i))}$ for their parity, and passes this "message" to the n-th bit node. Finally, the n-th bit node computes λ_n according to (20) by summing all of the incoming messages and adding $(2/\sigma^2) r_n$.

2.4. The Message-Passing Algorithm

But how to calculate $\lambda_{i,l}$? The key result is that $\lambda_{i,l}$ can be calculated *recursively*, again using an equation of the form (20), as long as the graph is cycle-free.

Consider the bit node labeled $c_{i,l}$ in Fig. 3. Obviously it is dependent on c_n, since both take part in the same parity-check equation. However, when conditioned on $\{r_{i \neq n}\}$, $c_{i,l}$ is *independent* of c_n. Even stronger, when we exclude the n-th observation r_n, we also exclude all of the observations that *pass through* r_n. This means that, for Fig. 3, two-thirds of the other

observations are excluded when r_n is excluded. Only the remaining third that are in the same subgraph as $c_{i,l}$ are relevant in computing $\lambda_{i,l}$. In effect, by excluding r_n we are cutting the graph on the edges leaving bit-node n, producing j disjoint graphs. Since the resulting graphs are disjoint, they can be treated independently, and since they are all cycle-free, we can directly apply (20) to calculate the a posteriori LLRs. This recursion may then progress through the tree, until the leaves of the tree are reached.

The *message-passing algorithm* is a decoding technique in which messages are passed from node to node through the Tanner graph. The nodes act as independent processors, collecting incoming messages and producing outgoing messages. There is no global control over the timing or the content of the messages; instead, the bit and check nodes follow a common *local* rule: Send a message as soon as all necessary incoming messages have been received. When the graph is cycle-free, the message-passing algorithm is a recursive algorithm that always converges to the true a posteriori log-likelihood ratios defined by (15) after a finite number of messages have been passed. However, most (if not all) "good" codes have cycles in their Tanner graphs. When applied to codes with cycles, the message-passing algorithm is no longer exact but approximate. Fortunately, even when the graph has cycles, the message-passing algorithm performs remarkably well, and its complexity is extremely low.

The message-passing decoder for a binary code with parity-check matrix **H** can be summarized concisely in terms of the index sets $\mathcal{M}_n = \{m\colon H_{m,n} = 1\}$ and $\mathcal{N}_m = \{n\colon H_{m,n} = 1\}$, as follows. Let $u_{m,n}^{(l)}$ denote an "upward" message from check-node m to bit-node n during the l-th iteration, and let $\lambda_n^{(l)}$ denote an estimate of the n-th a posteriori LLR (15) after l iterations. The message-passing decoder is:

Initialize —

- $u_{m,n}^{(0)} = 0$, for all $m \in \{1, \ldots M\}$ and $n \in \mathcal{N}_m$;
- $\lambda_n^{(0)} = (2/\sigma^2) r_n$, for all $n \in \{1, \ldots N\}$.

Iterate — for iteration counter $l = 1, 2, \ldots l_{\max}$**:**

- *(Check-node update)*
  for $m \in \{1, \ldots M\}$ and $n \in \mathcal{N}_m$:

$$u_{m,n}^{(l)} = -2\tanh^{-1}\left(\prod_{i \in \mathcal{N}_m - n} \tanh\left(\frac{-\lambda_i^{(l-1)} + u_{m,i}^{(l-1)}}{2} \right) \right) ; \tag{23}$$

- *(Bit-node update)*
  for $n \in \{1, \ldots N\}$:

$$\lambda_n^{(l)} = (2/\sigma^2) r_n + \sum_{m \in \mathcal{M}_n} u_{m,n}^{(l)} . \tag{24}$$

This algorithm can be interpreted in terms of passing messages through the Tanner graph. At the zero-th iteration, the message passed from the n-th bit node to each of its participating check nodes is $(2/\sigma^2) r_n$. The m-th check node collects its incoming messages, and passes as

an outgoing message the LLR for the parity of the bits involved in the m-th check, excluding the recipient. Each bit node receives j messages, and the n-th bit node produces a new estimate for λ_n by adding $(2/\sigma^2)r_n$ to the summation of all of the incoming messages, as dictated by (20). The process then repeats: the check nodes pass their new estimates for $\{\lambda_n\}$ to the check nodes, excluding the contribution from the recipient, and so on.

The message-passing view of the iterative decoder makes it possible to realize a parallel implementation based on the Tanner graph, in which each of the M check nodes is a separate processor, and each of the N bit nodes is simply a summing node. Such an implementation makes feasible the use of LDPC code and iterative decoding at extremely high bit rates. The memory requirements would be minimal, although layout of such a decoder would be difficult when N is large. At the other extreme is a fully serialized implementation, consisting of a single check-node processor that computes each of the Mk check-to-bit messages one by one. The memory requirements can be significant. A software implementation works essentially this way.

The serial decoder can be structured to precisely mimic the parallel decoder, but if some LLRs are updated before others, then the serial decoder will perform somewhat differently.

Exercise 3. Show that the Tanner graph for a (3, 1) repetition code has no cycles. Show that the message-passing algorithm converges to the true a posteriori LLRs after two iterations.

Example 3. Simulation results for the message-passing decoder of (23) and (24) are shown in Fig. 4, assuming the (3, 4) regular LDPC code of (3). The code rate is $R = 0.35$. After 1000 iterations, the code requires 6 dB to achieve a BER of 10^{-4}. In contrast, an uncoded BPSK system requires 8.4 dB to achieve the same BER; hence, this simple length-20 code offers a coding gain of 2.4 dB. However, the performance is far (about 6.5 dB) from the Shannon limit of $(2^{2R} - 1)/(2R) = -0.5$ dB for a code with rate $R = 0.35$. The figure shows that 30 iterations is enough to come within 0.2 dB of a receiver that performs 3000 iterations.

Example 4. Even though the Tanner graph for the (7, 4) Hamming code has cycles, the message-passing decoder is still effective. In Fig. 5, we show the BER performance after 7 iterations, and see an improvement of more than 1.5 dB when compared to a hard ML decoder.

3. Density Evolution

Analysis of the convergence properties of the message-passing decoder is difficult when the block length N is finite, but it simplifies considerably if we allow N to tend towards infinity. Roughly speaking, for a fixed iteration number l, the probability that a randomly chosen bit node from a randomly constructed LDPC graph is part of a cycle of length less than l tends towards zero as $N \to \infty$. This fact allows us to ignore cycles as $N \to \infty$, in which case the messages incident on any node will be independent.

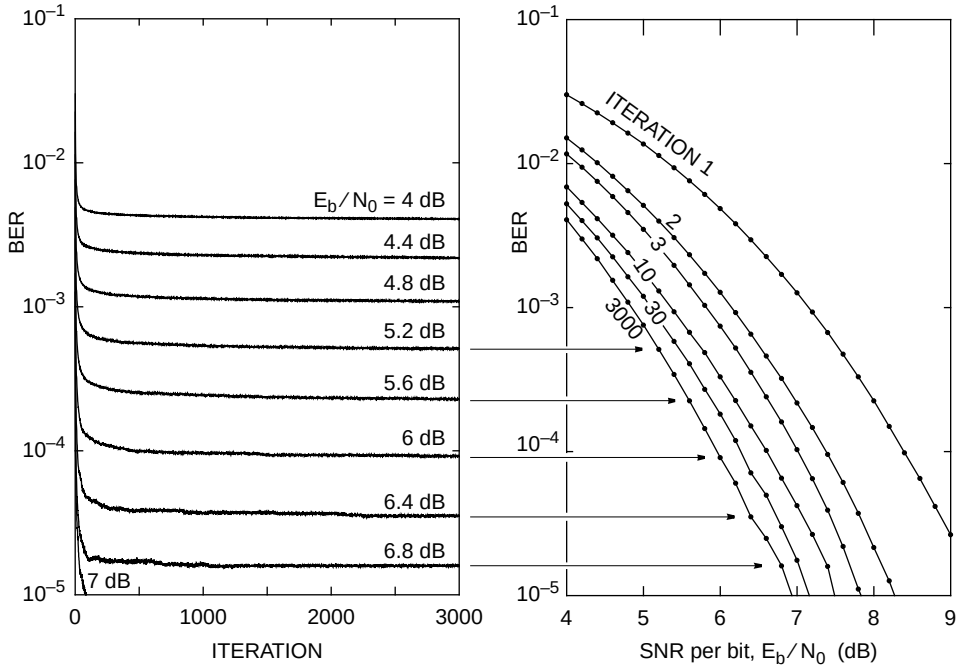

Fig. 4. Simulation results for the (3,4) regular LDPC code of (3), with blocklength 20 and rate R = 0.35, decoded using the message-passing algorithm of (23) and (24). These results were averaged over 6×10^5 message blocks (of $K = 7$ bits each).

Density evolution is a technique for tracking the pdfs of the messages in the Tanner graph of an LDPC code, under the assumption that $N \to \infty$. By symmetry of the code, the channel, and the decoding algorithm, we may assume without loss of generality that the all-zero codeword was transmitted. (We also assume a $0 \to 1$, $1 \to -1$ bit-to-symbol mapping in this section, in contrast to our prior discussion). Therefore, the initial message sent from the n-th bit node for the AWGN channel is:

$$\lambda_n^{(0)} = (2/\sigma^2)(1 + \mathcal{N}(0, \sigma^2)) \,. \tag{25}$$

Observe that the variance of $\lambda_n^{(0)}$ is twice its mean:

$$\lambda_n^{(0)} \sim \mathcal{N}(m, 2m) \,, \tag{26}$$

where $m = (2/\sigma^2)$. A Gaussian random variable with this property is said to be *consistent.*

Although the initial messages are Gaussian, subsequent messages are not. Nevertheless, to simplify analysis even further, it is convenient to assume that all messages have a *consistent* Gaussian pdf. In this way, the problem of tracking an infinite-dimensional pdf collapses to one of tracking only a single parameter: the mean. In the following we briefly summarize the

Gaussian approximation to density evolution [11][12][13][14]. We will assume that the LDPC code is regular with bit-node degree j and check-node degree k, although the analysis can be extended to irregular codes.

Let $u^{(l)}$ denote a randomly chosen upward (check-to-bit) message after l iterations, as calculated by a message-passing decoder using (23), and let μ_l denote its mean, $\mu_l = \mathrm{E}[u^{(l)}]$. Then, according to the tanh rule, this message satisfies:

$$\tanh\left(\frac{u^{(l)}}{2}\right) = \prod_{i=1}^{k-1} \tanh\left(\frac{v_i^{(l)}}{2}\right), \tag{27}$$

where $v_1{}^{(l)}, \ldots v_{k-1}{}^{(l)}$ are the relevant downward (bit-to-check) message after l iterations, each with mean $m_l = \mathrm{E}[v_i^{(l)}]$. We will assume that both $u^{(l)}$ and $v^{(l)}$ are consistent Gaussian random variables, so that $u^{(l)} \sim \mathcal{N}(\mu_l, 2\mu_l)$ and $v^{(l)} \sim \mathcal{N}(m_l, 2m_l)$. Taking the expectation of both sides of (27), and exploiting the independence of $\{v_i^{(l)}\}$, we have:

$$\psi(\mu_l) = \psi(m_l)^{k-1}, \tag{28}$$

where we have introduced the function:

$$\psi(m) = \mathrm{E}\left[\tanh(\tfrac{1}{2}\mathcal{N}(m, 2m))\right]. \tag{29}$$

This function is shown in Fig. 6, where it is seen to have behavior similar to $\tanh(m/2)$. In particular, we observe that the function is invertible.

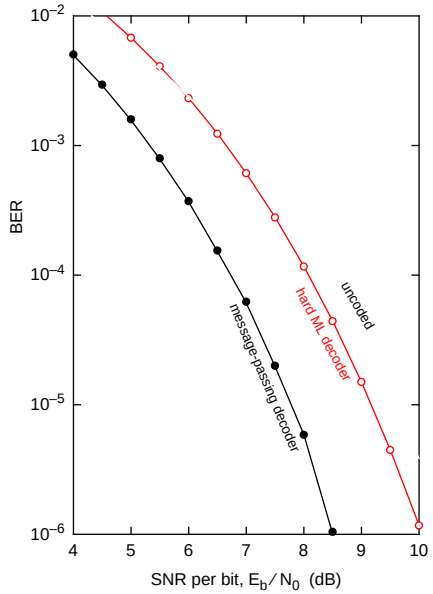

Fig. 5. Performance results for the (7, 4) Hamming code over an AWGN channel. The message-passing decoder outperforms the hard decoder by more than 1.5 dB after 7 iterations. (These results were averaged over 2×10^6 blocks of 4 message bits each.)

We now relate m_l of (28) to μ_{l-1}. The i-th downward message $v_i^{(l)}$ in (27) can be expressed as:

$$v_i^{(l)} = v_i^{(0)} + \sum_{n=1}^{j-1} u_n^{(l-1)}, \tag{30}$$

where $v_i^{(0)}$ is the *initial* downward message (which is a consistent Gaussian with mean $2/\sigma^2$), and where $u_1^{(l-1)}, \ldots u_{j-1}^{(l-1)}$ are the relevant upward messages impinging on the bit node, which are mutually independent, each with mean μ_{l-1}. Thus, averaging (30) yields:

$$m_l = 2/\sigma^2 + (j-1)\mu_{l-1}. \tag{31}$$

Substituting (31) into (28) yields

$$\psi(\mu_l) = \psi\Big(2/\sigma^2 + (j-1)\mu_{l-1}\Big)^{k-1}. \tag{32}$$

Inverting $\psi(\cdot)$ leads to the following recursive relationship for the mean μ_l of a randomly chosen upward message after l iterations:

$$\mu_l = \psi^{-1}\left(\psi\Big(2/\sigma^2 + (j-1)\mu_{l-1}\Big)^{k-1}\right). \tag{33}$$

As dictated by the message-passing decoder, the initial upward message is zero, so that $\mu_0 = 0$. With this initialization, (33) can be iterated to determine the evolution of μ_l. The dynamics of this recursion are completely determined by three parameters: the bit-node degree j, the check-node degree k, and the SNR through $2/\sigma^2$.

Example 5. Let $j = 4$, $k = 6$, and suppose we fix the SNR per bit at $E_b/N_0 = 1.72$ dB. (Recall that $E_b/N_0 = 1/(2R\sigma^2)$, where the rate is $R = 1 - j/k = 1/3$ in this example.) Then by iterating (33) we find that the mean μ_l approaches a constant of $\mu_l \to 0.375$. In Fig. 7, we plot (using a dashed line) the pdf of a consistent Gaussian pdf with mean 0.375. We see that there is a significant probability that the randomly chosen message will be negative, even after a large number of iterations. This implies that the probability of decoding error is nonzero.

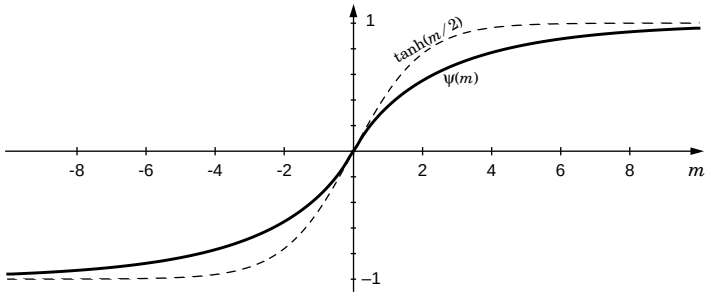

Fig. 6. The function $\psi(m)$ is roughly a stretched version of the function $\tanh(m/2)$.

Example 6. Suppose we again fix $j = 4$ and $k = 6$, but increase the SNR per bit from 1.72 dB to 1.73 dB. By iterating (33), we find that this small increase in SNR has a huge impact on μ_l, with $\mu_l \to \infty$ as $l \to \infty$. The solid curves in Fig. 7 show the pdf of a consistent Gaussian random variable with the corresponding mean μ_l for iteration $l \in \{630, 631, 632, 633, 634\}$. As the mean gets larger, the probability of a negative message diminishes, until an infinite mean implies a vanishingly small probability of decoding error.

For any set of values of j, k, and σ^2, the recursion (33) always converges. Furthermore, there exists a SNR *threshold* γ such that $\mu_l \to \infty$ as $l \to \infty$ for all $E_b/N_0 > \gamma$, indicating that the decoding error probability approaches zero, while μ_l converges to a finite number for all $E_b/N_0 < \gamma$ as $l \to \infty$, indicating that the decoding error probability is nonzero.

Example 7. Continuing the previous example, in Fig. 8 we plot the mean μ_l as a function of iteration l for different values of E_b/N_0, as found by iterating (33) with $j = 4$ and $k = 6$. The figure clearly illustrates the threshold phenomenon, with a threshold value near $\gamma = 1.73$ dB. The Shannon limit $(2^{2R} - 1)/(2R)$ is -0.55 dB for rate-$1/3$ codes. In contrast, the results of Fig. 8 suggest that a sufficiently long (4, 6) regular LDPC code can achieve vanishingly small error probability at $E_b/N_0 = 1.73$ dB. Hence, the gap to capacity is 2.28 dB. This gap can be further reduced using irregular LDPC codes [4][5].

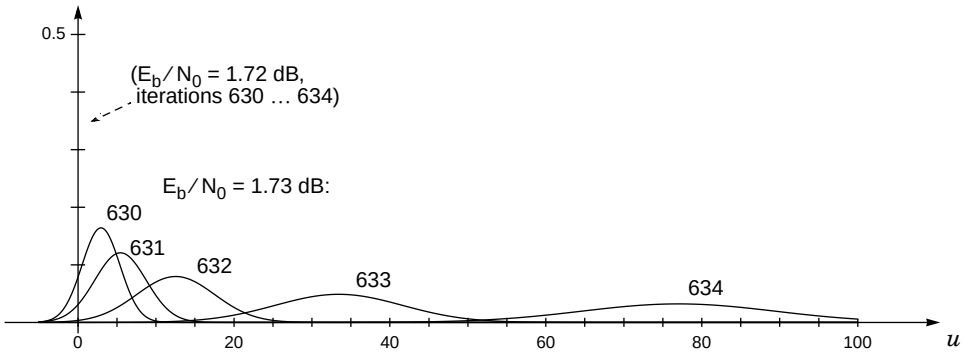

Fig. 7. The pdf of an upward message u as a function of E_b/N_0 and iteration.

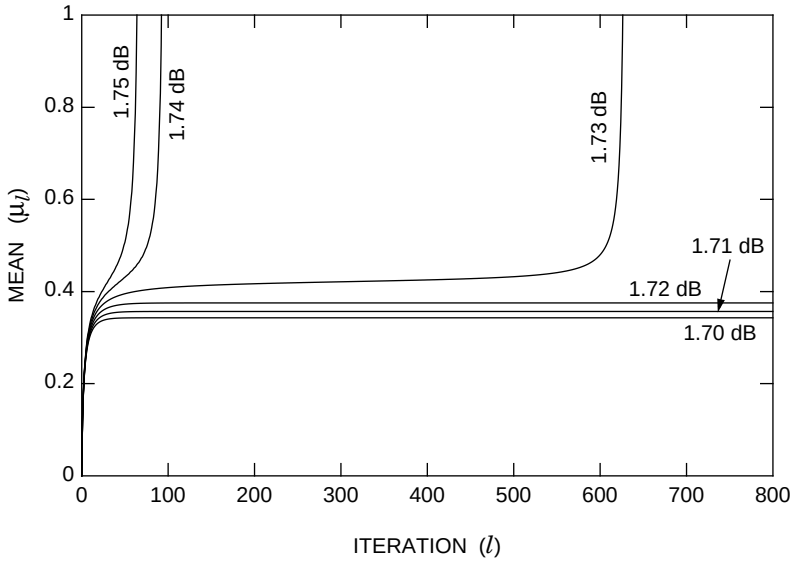

Fig. 8. The results of density evolution based on the Gaussian approximation for a (4, 6) regular LDPC code. Here, the mean of a randomly selected message is plotted as a function of iteration, for several different values of SNR. For E_b/N_0 = 1.73 dB and larger, the mean approach infinity, which implies that the error probability approaches zero.

Appendix A. Derivation of the Tanh Rule (7)

The parity $\phi(\boldsymbol{c})$ of a binary vector $\boldsymbol{c} = [c_1 \ldots c_n]$ can be expressed as:

$$\phi(\boldsymbol{c}) = \tfrac{1}{2}\Big(1 - \prod_{i=1}^{n}(1-2c_i)\Big), \tag{34}$$

since an even number of ones in $\boldsymbol{c}$ yields $\phi(\boldsymbol{c}) = 0$, while an odd number yields $\phi(\boldsymbol{c}) = 1$. The probability that the parity is odd is then the expected value of $\phi(\boldsymbol{c})$:

$$\Pr[\phi(\boldsymbol{c}) = 1] = \mathrm{E}[\phi(\boldsymbol{c})] \tag{35}$$

$$= \tfrac{1}{2}\Big(1 - \mathrm{E}\Big[\prod_{i=1}^{n}(1-2c_i)\Big]\Big) \tag{36}$$

$$= \tfrac{1}{2}\Big(1 - \prod_{i=1}^{n}(1-2\mathrm{E}[c_i])\Big) \quad \text{(because of independence)} \tag{37}$$

$$= \tfrac{1}{2}\Big(1 - \prod_{i=1}^{n}\big(1 - \frac{2e^{\lambda_i}}{1+e^{\lambda_i}}\big)\Big) \tag{38}$$

$$= \tfrac{1}{2}\Big(1 - \prod_{i=1}^{n}\frac{1-e^{\lambda_i}}{1+e^{\lambda_i}}\Big) \tag{39}$$

$$= \tfrac{1}{2}\Big(1 - \prod_{i=1}^{n}\tanh(-\lambda_i/2)\Big). \tag{40}$$

Since $\Pr[\phi(\boldsymbol{c}) = 0] = 1 - \Pr[\phi(\boldsymbol{c}) = 1]$, the LLR for $\phi(\boldsymbol{c})$ becomes:

$$\lambda_{\phi(\boldsymbol{c})} = \log\frac{\Pr[\phi(\boldsymbol{c}) = 1]}{\Pr[\phi(\boldsymbol{c}) = 0]} \tag{41}$$

$$= \log\left(\frac{1 - \Pi_i\tanh(-\lambda_i/2)}{1 + \Pi_i\tanh(-\lambda_i/2)}\right). \tag{42}$$

Using $\tanh(-\lambda/2) = (1 - e^{\lambda})/(1 + e^{\lambda})$, and letting $\Pi = \prod_{i=1}^{n}\tanh(-\lambda_i/2)$, we get:

$$\tanh(-\lambda_{\phi(\boldsymbol{c})}/2) = \frac{1 - \left(\frac{1-\Pi}{1+\Pi}\right)}{1 + \left(\frac{1-\Pi}{1+\Pi}\right)} = \frac{(1+\Pi) - (1-\Pi)}{(1+\Pi) + (1-\Pi)} = \Pi = \prod_{i=1}^{n}\tanh(-\lambda_i/2), \tag{43}$$

which proves the tanh rule of (7).

Appendix B. Derivation of Additive Rule (9)

Substituting $-\lambda_i = \mathrm{sign}(-\lambda_i)\,|\lambda_i|$ into (7) yields the pair of equations:

$$\mathrm{sign}(-\lambda_{\phi(\boldsymbol{c})}) = \prod_{i=1}^{n}\mathrm{sign}(-\lambda_i)\,, \tag{44}$$

$$\tanh\Big(\frac{|\lambda_{\phi(\boldsymbol{c})}|}{2}\Big) = \prod_{i=1}^{n}\tanh\Big(\frac{|\lambda_i|}{2}\Big)\,. \tag{45}$$

Taking $-\log(\cdot)$ of both sides of (45) yields:

$$f(|\lambda_{\phi(\boldsymbol{c})}|) = \sum_{i=1}^{n} f(|\lambda_i|)\,, \tag{46}$$

where $f(x)$ is defined by (10). Because $f(f(x)) = x$ for all $x > 0$, we can apply $f(\cdot)$ to both sides of (46), yielding:

$$|\lambda_{\phi(\boldsymbol{c})}| = f\left(\sum_{i=1}^{n} f(|\lambda_i|)\right). \tag{47}$$

Combining (44) and (47) yields (9).

References

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, January 1962.

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

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

[4] S.-Y. Chung, G. D. Forney, Jr., T. J. 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, February 2001.

[5] T. J. Richardson, A. Shokrollahi, and R. Urbanke, "Design of Capacity-Approaching Low-Density Parity-Check Codes, *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.

[6] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July/Oct. 1948.

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

[8] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498-519, February 2001.

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429–445, March 1996.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *Proc. IEEE Int. Conf. Communications*, pages 1064–1070, Geneva, May 1993.

[11] S.-Y. Chung, R. Urbanke, and T. J. Richardson, "Gaussian Approximation for Sum-Product Decoding of Low-Density Parity-Check Codes," in *Proc. Int. Symp. Inform. Theory*, page 318, Sorrento, Italy, June 2000.

[12] S.-Y. Chung, "On the Construction of Some Capacity-Approaching Coding Schemes," *Ph.D. Dissertation*, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[13] S.-Y. Chung, T. J. Richardson, and R. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes using a Gaussian Approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657-670, February 2001.

[14] H. El Gamal and A. R. Hammons, Jr., "Analyzing the Turbo Decoder using the Gaussian Approximation," in *Proc. Int. Symp. Inform. Theory*, page 319, Sorrento, Italy, June 2000.