Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation

Sae-Young Chung, *Member, IEEE*, Thomas J. Richardson, and Rüdiger L. Urbanke

***Abstract*—Density evolution is an algorithm for computing the capacity of low-density parity-check (LDPC) codes under message-passing decoding. For memoryless binary-input continuous-output additive white Gaussian noise (AWGN) channels and sum-product decoders, we use a Gaussian approximation for message densities under density evolution to simplify the analysis of the decoding algorithm. We convert the infinite-dimensional problem of iteratively calculating message densities, which is needed to find the exact threshold, to a one-dimensional problem of updating means of Gaussian densities. This simplification not only allows us to calculate the threshold quickly and to understand the behavior of the decoder better, but also makes it easier to design good irregular LDPC codes for AWGN channels.**

For various regular LDPC codes we have examined, thresholds can be estimated within 0.1 dB of the exact value. For rates between 0.5 and 0.9, codes designed using the Gaussian approximation perform within 0.02 dB of the best performing codes found so far by using density evolution when the maximum variable degree is 10. We show that by using the Gaussian approximation, we can visualize the sum-product decoding algorithm. We also show that the optimization of degree distributions can be understood and done graphically using the visualization.

***Index Terms*—Density evolution, fixed points, Gaussian approximation, low-density parity-check (LDPC) codes, stability, sum-product algorithm, threshold.**

I. Introduction

FOR many channels and iterative decoders of interest, low-density parity-check (LDPC) codes—first discovered by Gallager [1], [2] and rediscovered by Spielman *et al.* [3] and MacKay *et al.* [4], [5]—exhibit a threshold phenomenon: as the block length tends to infinity, an arbitrarily small bit-error probability can be achieved if the noise level is smaller than a certain threshold. For a noise level above this threshold, on the other hand, the probability of bit error is larger than a positive constant. Gallager first observed this phenomenon for binary symmetrical channels (BSCs) in his introduction of regular LDPC codes [1], [2], where he used an explicit construction of regular graphs. Luby *et al.* generalized this idea to randomly constructed irregular LDPC codes, showed that irregular codes perform better than regular ones, and also showed that the threshold phenomenon occurs for these codes [6].

In [7], this observation was further generalized by Richardson and Urbanke to a large range of binary-input channels, including binary erasure, binary symmetric, Laplace, and AWGN channels, and to various decoding algorithms including belief propagation (sum-product algorithm), which are collectively called *message-passing* algorithms. Richardson *et al.* proved a general concentration result showing that the decoder performance on random graphs converges to its expected value as the length of the code increases, generalizing the result of Luby *et al.* [7]. Since it is difficult to determine this expected performance for an ensemble of finite size, they used the expected behavior in the limit of infinitely long codes, which can be determined from the corresponding cycle-free graph. They defined the threshold as indicated above for a random ensemble of irregular codes specified by degree distributions, and developed an algorithm called *density evolution* for iteratively calculating message densities, enabling the determination of thresholds.

Using this result, they constructed LDPC codes that clearly beat the powerful turbo codes [8] on AWGN channels. Recently, this was improved [9], [10], where the threshold for a rate-$1/2$ LDPC code on the AWGN channel is within 0.0045 dB of the Shannon limit and simulation results are within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} using a block length of 10^7.

Calculating thresholds and optimizing degree distributions using density evolution are computationally intensive tasks, and are often difficult for most channels other than binary-erasure channels (BECs). In BECs, density evolution becomes one-dimensional, and it is possible to do more analysis and even to construct capacity-achieving codes [11], [30]. For more interesting channels including AWGN channels, however, density evolution is too complicated to be analyzed.

In this paper, we present a simple method to estimate the threshold for irregular LDPC codes on memoryless binary-input continuous-output AWGN channels with sum-product decoding. This method is based on approximating message densities as Gaussians (for regular LDPC codes) or Gaussian mixtures (for irregular LDPC codes) [12]. We show that, without much sacrifice in accuracy, a one-dimensional quantity, namely, the mean of a Gaussian density, can act as faithful surrogate for the message density, which is an infinite-dimensional vector.

Manuscript received January 7, 2000; revised July 10, 2000. This work was supported in part by Lucent Technologies. The material in this paper was presented in part at the 2000 International Symposium of Information Theory, Sorrento, Italy, June 25–30, 2000.

S.-Y. Chung was with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with Airvana, Inc., Chelmsford, MA 01824 USA (e-mail: sae-young.chung@airvananet.com).

T. J. Richardson was with Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 USA. He is now with Flarion Technologies, Bedminster, NJ 07921 USA (e-mail: richardson@flarion.com).

R. L. Urbanke was with Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 USA. He is now with the Communications Theory Lab, EPFL, DSC-LTHC, CH-1015 Lausanne, Switzerland (e-mail: Rudiger.Urbanke@epfl.ch).

Communicated by B. J. Frey, Guest Editor.

Publisher Item Identifier S 0018-9448(01)00717-9.

Since this method is easier to analyze and computationally faster than density evolution, it can be a useful tool for understanding the behavior of the decoder and for optimizing irregular codes. For example, we show how to determine the rate of convergence of the error probability, and why there is an alternation between fast and slow decoding stages, as noted in [13]. We also use this method to find good irregular codes using linear programming. This algorithm not only optimizes degree distributions several orders of magnitude faster, but it is often as good as the optimization methods based on the full density evolution.

For turbo codes, a one-dimensional Gaussian approximation based on extrinsic information transfer chart (EXIT chart) was first used by ten Brink [14], where he considered parallel concatenated codes. Using his EXIT chart methods, he recently designed a rate-$1/2$ serial concatenated code with a repetition code and a eight-state rate-one code that is within about 0.1 dB of the Shannon limit [15]. A similar Gaussian-approximation method based on signal-to-noise ratio (SNR) was later used in [16] for turbo codes, which appeared at the same time as [12]. See also [17] in this issue. Divsalar *et al.* also used a similar approximation method based on SNR [18].

Since there is no simple formula for updating message densities for turbo decoding, Monte Carlo simulations were used to analyze approximate trajectories for Gaussian messages under turbo decoding in both papers [14], [16]. Similar to the mean as used in this paper, they used one-dimensional quantities, namely, mutual information [14] and SNR [16], to approximate message densities. They showed that the Gaussian approximations work reasonably well for turbo decoding, and argued that the position of the turbo waterfall region can be estimated by visualizing the trajectories.

In other related works, the sum-product algorithm was analyzed for graphs with cycles when messages are jointly Gaussian [19]–[21], [31]. Since a Gaussian density is completely characterized by its mean vector and covariance matrix, the analysis of the sum-product algorithm becomes tractable. The main purpose of these works is to analyze how well decoding works on graphs with cycles. Surprisingly, the means converge to the correct posterior means when certain conditions are satisfied, even when there are cycles in the graph.

Let us first consider a regular binary (d_v, d_c)-LDPC code, where d_v denotes the number of neighbors of a variable node and d_c denotes the number of neighbors of a check node. Under the sum-product algorithm, and message passing in general, variable nodes and check nodes exchange messages iteratively. A check node gets messages from its d_c neighbors, processes the messages, and sends the resulting messages back to its neighbors. Similarly, a variable node receives messages from its d_v neighbors, processes the messages, and sends messages back to its neighbors. Each output message of a variable or a check node is a function of all incoming messages to the node except the incoming message on the edge where the output message will be sent out. This restriction is essential for the sum-product algorithm to produce correct marginal *a posteriori* probabilities for cycle-free graphs.

This two-step procedure is repeated many times. After n such iterations, the variable node decodes its associated bit based on all information obtained from its depth-n subgraph of neighbors. Under the "local tree assumption," namely, that the girth of the graph is large enough so that the subgraph forms a tree (i.e., there are no repeated nodes in the subgraph), we can analyze the decoding algorithm straightforwardly because incoming messages to every node are independent. Furthermore, by the general concentration theorem of [6], which generalizes the results of [7], we are assured that, for almost all randomly constructed codes and for almost all inputs, the decoder performance will be close to the decoder performance under the local tree assumption with high probability, if the block length of the code is long enough. From now on we base our analysis on this local tree assumption.

It is convenient to use log-likelihood ratios (LLRs) as messages; i.e., we use

$$v = \log \frac{p(y|x=1)}{p(y|x=-1)}$$

as the output message of a variable node, where x is the bit value of the node and y denotes all the information available to the node up to the present iteration obtained from edges other than the one carrying v. Likewise, we define the output message of a check node as

$$u = \log \frac{p(y'|x'=1)}{p(y'|x'=-1)}$$

where x' is the bit value of the variable node that gets the message from the check node, and y' denotes all the information available to the check node up to the present iteration obtained from edges other than the one carrying u.

Let v be a message from a variable node to a check node. Under sum-product decoding, v is equal to the sum of all incoming LLRs; i.e.,

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

where u_i, $i = 1, \dots, d_v - 1$, are the incoming LLRs from the neighbors of the variable node except the check node that gets the message v, and u_0 is the observed LLR of the output bit associated with the variable node. The density of the sum of d_v random variables u_i, $i = 0, \dots, d_v - 1$, can be easily calculated by convolution of densities of the u_i's, which can be efficiently done in the Fourier domain [6].

The message update rule for check nodes can be obtained from the duality between variable and check nodes and the resulting Fourier transform relationship between (p, q) and $(p + q, p - q)$, where $p = p(x = 1|y)$ and $q = p(x = -1|y)$ [22]. From this, we get the following "tanh rule" [23]–[25], [6], [22]:

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

where v_j, $j = 1, \dots, d_c - 1$, are the incoming LLRs from $d_c - 1$ neighbors of a check node, and u is the message sent to the remaining neighbor. Figs. 1 and 2 show message flows through a variable node and a check node, respectively, where we use normal realizations [22] of variable and check nodes, which be-

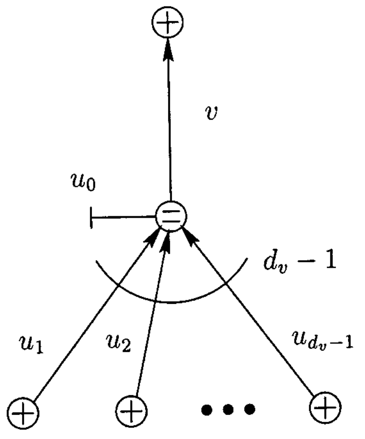

Fig. 1. Message flow through a variable node.

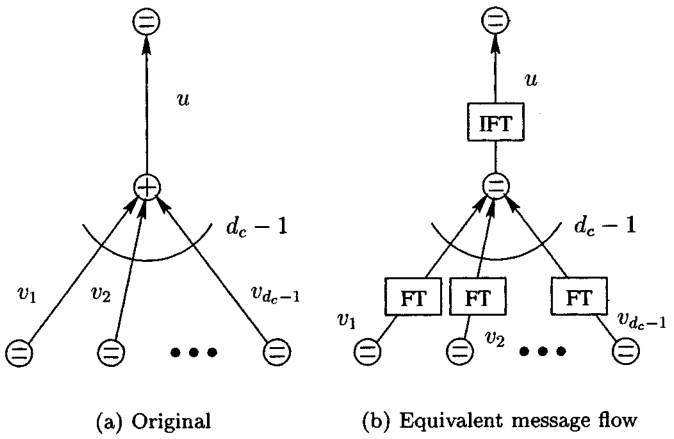

Fig. 2. Message flow through a check node.

come a repetition and a parity check, respectively. In this representation, half-edges represent variables and full edges represent states. Under sum-product decoding, constraints become computation nodes and states become communication links between constraint nodes. Fig. 2(b) shows how the decoding for a check node can be done on a dual graph (repetition code) using Fourier transforms, which is a special case of a more general idea of dualizing the sum-product algorithm on a dual graph [22].

To apply the same Fourier transform technique to (2), we need to take logarithms on each side to convert the product into a sum, i.e.

$$\left(s_u, \log\left|\tanh\frac{u}{2}\right|\right) = \sum_{j=1}^{d_c-1}\left(s_{v_j}, \log\left|\tanh\frac{v_j}{2}\right|\right) \tag{3}$$

where s_u and s_{v_j} are the signs of u and v_j, $1 \le j \le d_c - 1$, respectively. We define the sign s_x as 0 if $x \ge 0$ and 1 otherwise. Thus, the sum for the sign in (3) is performed in $\mathbb{Z}_2$ and the sum for the magnitude is the ordinary sum in $\mathbb{R}$. As in the first case, density evolution for this step can be done numerically in the Fourier domain [6].

This can be generalized to irregular codes, where a variable number of degrees is allowed. Let

$$\lambda(x) = \sum_{i=2}^{d_l} \lambda_i x^{i-1}$$

and

$$\rho(x) = \sum_{i=2}^{d_r} \rho_i x^{i-1}$$

be the generating functions of the degree distributions for the variable and check nodes, respectively, where λ_i and ρ_i are the fractions of edges belonging to degree-i variable and check nodes, respectively [11]. Using this expression, the nominal rate r of the code is given by

$$r = 1 - \frac{\int_0^1 \rho(x)\,dx}{\int_0^1 \lambda(x)\,dx}$$

[11]. For detailed density evolution for irregular codes, we refer readers to [13].

This two-stage computation, called density evolution, is the key step in calculating the thresholds of message-passing decoding in [6]. First, without loss of generality, we assume that the all-0 codeword was sent. Then, we fix the channel parameter, namely, noise power, and we run the above algorithm iteratively until either the density of v tends to the "point mass at infinity" (equivalently, the probability of error tends to zero), or it converges to a density with a finite probability of error (the probability of v being negative).[1] The threshold is defined as the maximum noise level such that the probability of error tends to zero as the number of iterations tends to infinity.[2]

II. Gaussian Approximation for Regular LDPC Codes

The LLR message u_0 from the channel is Gaussian with mean $2/\sigma_n^2$ and variance $4/\sigma_n^2$, where σ_n^2 is the variance of the channel noise. Thus, if all $\{u_i, i \ge 1\}$ [which are independent and identically distributed (i.i.d.)] are Gaussian in (1), then the resulting sum is also Gaussian because it is the sum of independent Gaussian random variables. Even if the inputs are not Gaussian, by the central limit theorem, the sum would look like a Gaussian if many independent random variables are added as noted in [26].

Indeed, if we normalize the output of (1) to a zero-mean unit-variance variable, and if the $\{u_i, i \ge 1\}$ are i.i.d. with finite variance, then the normalized output converges to a Gaussian distribution as d_v tends to infinity by the central limit theorem. However, due to the interaction between variable and check nodes, it does not seem easy to rigorously show how close the distributions are to the Gaussian in actual density evolution.

Using simulations, Wiberg [26] observed that message distributions for AWGN channels resemble Gaussians using a (2, 3)-regular code. Our empirical results using density evolution show that for a regular graph, the output v in (1) as well as the output u in (2) can be well approximated by Gaussian densities, although u tends to be less like Gaussian, especially when its mean is near zero. Despite this difference, the approximation

[1]A point mass at 0 contributes half of its probability mass to the probability of error.

[2]In the limit of infinitely long codes.

method that we develop in this section works very well for regular LDPC codes. From this point on, for regular LDPC codes, we assume that the variables u, v, u_i, v_j's are Gaussian.

Since a Gaussian is completely specified by its mean and variance, we need to keep only the means and variances during iterations. There is an important condition, called the *symmetry condition* [13], that is preserved under density evolution for all messages, which can be expressed as $f(x) = f(-x)e^x$, where $f(x)$ is the density of an LLR message. By enforcing this condition for the approximate Gaussian densities at every iteration, we can greatly improve the accuracy of the approximation. For a Gaussian with mean m and variance σ^2, this condition reduces to $\sigma^2 = 2m$, which means that we need to keep only the mean.

In [16], the SNR of a Gaussian was used instead of its mean, to approximate density evolution for turbo codes. If we define the SNR of a Gaussian with mean m and variance σ^2 as m^2/σ^2, then it becomes $m/2$ if we assume the symmetry condition for the approximate messages. Therefore, the SNR becomes equivalent to the mean. However, since simulations were used in [16] due to a lack of tools for analyzing the density evolution for turbo codes, the authors of [16] were unable to calculate thresholds for turbo codes with enough precision (up to two digits).

Since we are concentrating only on regular and irregular LDPC codes, we can find analytic expressions for the approximate density evolution and we can calculate the approximate threshold values with arbitrary precision. Our empirical results show that it is limited to about ten digits due to the precision of the double precision numbers. Using the mean is also intuitive and physically motivated especially for irregular codes. It needs more steps to show how different SNRs are mixed for irregular codes, which can be done naturally using the mean as we will show in the next section.

We denote the means of u and v by m_u and m_v, respectively. Then (1) simply becomes

$$m_v^{(\ell)} = m_{u_0} + (d_v - 1)m_u^{(\ell-1)} \tag{4}$$

where m_{u_0} is the mean of u_0 and ℓ denotes the ℓth iteration. We omit the index i because the u_i's are i.i.d. for $1 \le i < d_v$, and have the same mean m_u. Note that $m_u^{(0)} = 0$ since the initial message from any check node is 0.

The updated mean $m_u^{(\ell)}$ at the ℓth iteration can be calculated by taking expectations on each side of (2), i.e.

$$E\left[\tanh\frac{u^{(\ell)}}{2}\right] = E\left[\tanh\frac{v^{(\ell)}}{2}\right]^{d_c-1} \tag{5}$$

where we have omitted the index j and simplified the product because the v_j's are i.i.d. One complete iteration begins at the variable nodes and then ends at the check nodes. The variables $u^{(\ell)}$ and $v^{(\ell)}$ are Gaussian $\mathcal{N}(m_u^{(\ell)}, 2m_u^{(\ell)})$, and $\mathcal{N}(m_v^{(\ell)}, 2m_v^{(\ell)})$, respectively, where $\mathcal{N}(m, \sigma^2)$ is a Gaussian density with mean m and variance σ^2.

Note that the expectation $E[\tanh\frac{u}{2}]$ depends only on the mean m_u of u, since u is Gaussian with mean m_u and variance $2m_u$; i.e.

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

We define the following function $\phi(x)$ for $x \in [0, \infty)$, which will be useful and convenient for further analyses.

Definition 1:

$$\phi(x) = \begin{cases} 1 - \dfrac{1}{\sqrt{4\pi x}}\displaystyle\int_{\mathbb{R}} \tanh\frac{u}{2} e^{-\frac{(u-x)^2}{4x}}\,du, & \text{if } x > 0 \\ 1, & \text{if } x = 0. \end{cases}$$

It is easy to check that $\phi(x)$ is continuous and monotonically decreasing on $[0, \infty)$, with $\phi(0) = 1$ and $\phi(\infty) = 0$. Using $\phi(x)$, we obtain the following update rule for m_u from (4) and (5):

$$m_u^{(\ell)} = \phi^{-1}\left(1 - \left[1 - \phi\left(m_{u_0} + (d_v - 1)m_u^{(\ell-1)}\right)\right]^{d_c-1}\right) \tag{6}$$

where $m_u^{(0)} = 0$ is the initial value for m_u.

The following lemma gives upper and lower bounds on $\phi(x)$.

Lemma 1:

$$\sqrt{\frac{\pi}{x}}e^{-\frac{x}{4}}\left(1 - \frac{3}{x}\right) < \phi(x) < \sqrt{\frac{\pi}{x}}e^{-\frac{x}{4}}\left(1 + \frac{1}{7x}\right),\ x > 0. \tag{7}$$

Proof: By expanding $\frac{1}{1+e^u}$ as follows:

$$\frac{1}{1+e^u} = \begin{cases} \sum_{k=0}^{\infty} (-1)^k e^{ku}, & \text{if } u < 0 \\ \sum_{k=1}^{\infty} (-1)^{k-1} e^{-ku}, & \text{if } u > 0 \end{cases}$$

we obtain

$$\begin{aligned} \phi(x) &= \frac{1}{\sqrt{\pi x}}\int_{-\infty}^{0} \sum_{k=0}^{\infty} (-1)^k e^{ku} e^{-(u-x)^2/4x}\,du \\ &\quad + \frac{1}{\sqrt{\pi x}}\int_{0}^{\infty} \sum_{k=1}^{\infty} (-1)^{k-1} e^{-ku} e^{-(u-x)^2/4x}\,du \\ &= \frac{2}{\sqrt{\pi x}} e^{-x/4}\int_{0}^{\infty} \sum_{k=0}^{\infty} (-1)^k e^{-(\frac{1}{2}+k)u} e^{-u^2/4x}\,du \\ &= 4\sum_{k=0}^{\infty} (-1)^k e^{x(k^2+k)} Q\left(\sqrt{\frac{x}{2}}(1+2k)\right) \end{aligned}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-t^2/2}\,dt$$

is the Q-function. Using

$$\left(\frac{1}{x} - \frac{1}{x^3}\right)\frac{1}{\sqrt{2\pi}}e^{-x^2/2} < Q(x) < \left(\frac{1}{x}\right)\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$$

[27], we get

$$\begin{aligned} \phi(x) &< \frac{4e^{-x/4}}{\sqrt{\pi x}}\left[\sum_{k=0}^{\infty}\frac{(-1)^k}{1+2k} + \sum_{j=1}^{\infty}\frac{2}{x(4j-1)^3}\right] \\ &< \sqrt{\frac{\pi}{x}}e^{-x/4}\left(1 + \frac{1}{7x}\right) \end{aligned}$$

where we use $\sum_{k=0}^{\infty}\frac{(-1)^k}{1+2k} = \frac{\pi}{4}$ [28] and $\sum_{j=1}^{\infty}\frac{1}{(4j-1)^3} < \frac{\pi}{56}$. Similarly, we get

$$\begin{aligned} \phi(x) &> \frac{4e^{-x/4}}{\sqrt{\pi x}}\left[\sum_{k=0}^{\infty}\frac{(-1)^k}{1+2k} - \sum_{j=1}^{\infty}\frac{2}{x(4j-3)^3}\right] \\ &> \sqrt{\frac{\pi}{x}}e^{-x/4}\left(1 - \frac{3}{x}\right) \end{aligned}$$

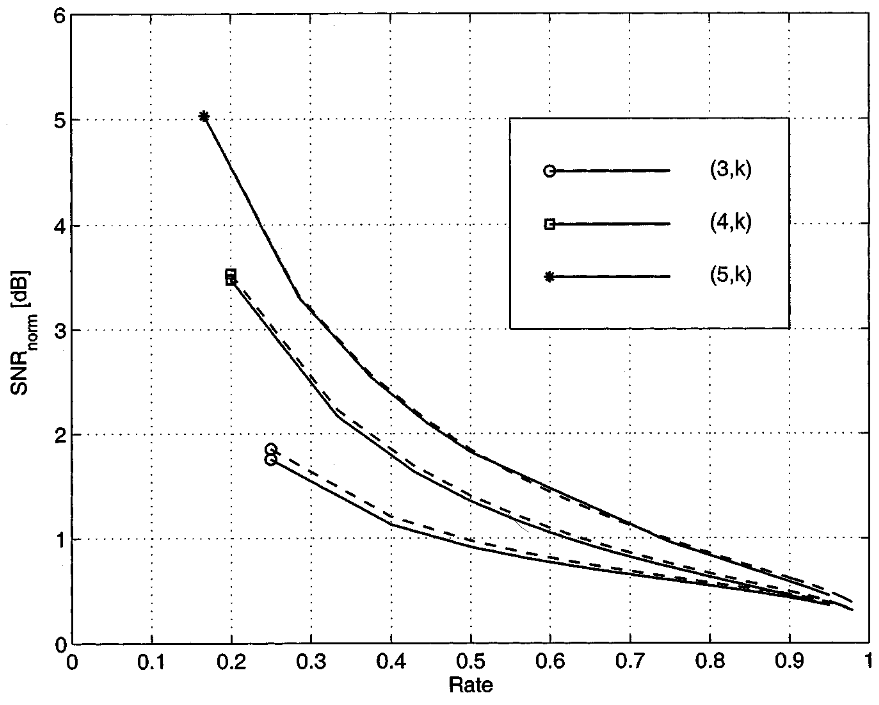

Fig. 3. Approximate (- -) and exact (—) thresholds for (j, k)-regular LDPC codes.

where we use

$$\sum_{j=1}^{\infty} \frac{1}{(4j-3)^3} < \frac{3\pi}{8}. \qquad \square$$

These bounds, which are tight as $x \to \infty$, will be used in analyzing the behavior of the decoder and in optimizing degree distributions in the following sections.

Although it is possible to calculate thresholds and to optimize degree distributions using the exact values of $\phi(x)$, we note that the following two approximations can be used instead to speed up the calculations without much sacrifice in accuracy. For small x, say $x < 10$, we will use the following approximation, which is very accurate and is better than (7) for this range of x:

$$\phi(x) \sim e^{\alpha x^{\gamma} + \beta} \tag{8}$$

where $\alpha = -0.4527$, $\beta = 0.0218$, and $\gamma = 0.86$ [these values were optimized to minimize the maximum absolute error between (8) and $\phi(x)$]. We note that there should be other curves that fit $\phi(x)$ better than (8), but (8) seems to be good enough for our purposes. For large x, say $x > 10$, we use the average of the upper and lower bounds of (7) as an approximation for $\phi(x)$.

Table I shows approximate thresholds σ_{GA} calculated using (6) compared to the exact thresholds σ_{exact} from density evolution. There is about 0.3 to 1.2% error (0.03 to 0.1 dB) compared to the exact values.

Fig. 3 shows differences between approximate and exact thresholds for various regular LDPC codes, where SNR$_{\text{norm}}$ is defined as the distance from the Shannon limit in decibels. Note that the accuracy for $(j, j+1)$ codes becomes improved as j becomes large, which can be explained because as more inputs are added at a variable node, the output becomes more Gaussian. Algorithms developed in [9] and [10] were used to calculate exact thresholds in Table I.

TABLE I
APPROXIMATE AND EXACT THRESHOLD VALUES FOR VARIOUS (j, k)-REGULAR LDPC CODES FOR THE BINARY-INPUT AWGN CHANNEL AND SUM-PRODUCT DECODING

j	k	rate	σ_{GA}	σ_{exact}	error[dB]
3	6	0.5	0.8747	0.8809	0.06
4	8	0.5	0.8323	0.8376	0.06
5	10	0.5	0.7910	0.7936	0.03
3	5	0.4	1.0003	1.0093	0.08
4	6	1/3	1.0035	1.0109	0.06
3	4	0.25	1.2517	1.2667	0.10
4	10	0.6	0.7440	0.7481	0.05
3	9	2/3	0.7051	0.7082	0.04
3	12	0.75	0.6297	0.6320	0.03

III. Gaussian Approximation for Irregular LDPC Codes

The preceding analysis can be easily extended to irregular LDPC codes.

We again consider an ensemble of random codes with degree distributions $\lambda(x)$ and $\rho(x)$. A node will thus get i.i.d. messages from its neighbors, where each of these messages is a random mixture of different densities from neighbors with different degrees. We denote these mixtures from variable nodes and from check nodes by v and u, respectively.

We assume first that the individual output of a variable or a check node is Gaussian, as in the preceing section, based on the empirical evidence. Therefore, the mean $m_{v,i}^{(\ell)}$ of the output message of a degree-i variable node at the ℓth iteration is given by

$$m_{v,i}^{(\ell)} = m_{u_0} + (i-1)m_u^{(\ell-1)}$$

where m_{u_0} is the mean of u_0 and $m_u^{(\ell-1)}$ is the mean of u at the $(\ell-1)$st iteration, where u is a Gaussian mixture in general [when $\rho(x)$ is not concentrated in one degree]. The variance of the output density is given by $2m_{v,i}^{(\ell)}$, as in the last section. Therefore, at the ℓth iteration, an incoming message v to a check node will have the following Gaussian mixture density $f_v^{(\ell)}$:

$$f_v^{(\ell)} = \sum_{i=2}^{d_l} \lambda_i \mathcal{N}(m_{v,i}^{(\ell)}, 2m_{v,i}^{(\ell)}) \tag{9}$$

where $m_{v,i}^{(\ell)}$ is the mean of the Gaussian output from a degree-i variable node.

Using (9), we get

$$E\left[\tanh\frac{v^{(\ell)}}{2}\right] = 1 - \sum_{i=2}^{d_l} \lambda_i \phi(m_{v,i}^{(\ell)}).$$

Then, we use (5) for a check node to calculate the mean of its output. Therefore, the mean $m_{u,j}^{(\ell)}$ of the Gaussian output message $u_j^{(\ell)}$ of a check node with degree j at the ℓth iteration is given by

$$m_{u,j}^{(\ell)} = \phi^{-1}\left(1 - \left[1 - \sum_{i=2}^{d_l} \lambda_i \phi(m_{v,i}^{(\ell)})\right]^{j-1}\right).$$

Since $u = u_j$ with probability ρ_j for $2 \le j \le d_r$, u is also a Gaussian mixture. Since the output message of a variable node is characterized by its mean, which is the sum of the means of the incoming densities, we need to keep only the mean of the Gaussian mixture from check nodes to variable nodes. By linearly combining these means for degree-2, $\ldots$, d_r check nodes with weights $\{\rho_i, 2 \le i \le d_r\}$, we get $m_u^{(\ell)}$ as given in the following:

$$m_u^{(\ell)} = \sum_{j=2}^{d_r} \rho_j \phi^{-1}\left(1 - \left[1 - \sum_{i=2}^{d_l} \lambda_i \phi\left(m_{u_0} + (i-1)m_u^{(\ell-1)}\right)\right]^{j-1}\right). \tag{10}$$

For $0 < s < \infty$ and $0 \le t < \infty$, we define $f_j(s, t)$ and $f(s, t)$ as

$$\begin{aligned} f_j(s,t) &= \phi^{-1}\left(1 - \left[1 - \sum_{i=2}^{d_l} \lambda_i \phi(s + (i-1)t)\right]^{j-1}\right) \\ f(s,t) &= \sum_{j=2}^{d_r} \rho_j f_j(s,t). \end{aligned} \tag{11}$$

Using this, we rewrite (10) as

$$t_\ell = f(s, t_{\ell-1}) \tag{12}$$

where $s = m_{u_0}$ and $t_\ell = m_u^{(\ell)}$. The initial value t_0 is 0. Note that since $t_1 = f(s, 0) > 0$ for $s > 0$, the iteration (12) will always start.

Definition 2: The threshold s^* is the infimum of all s in $\mathbb{R}^+$ such that $t_\ell(s)$ converges to ∞ as $\ell \to \infty$.

The threshold in terms of noise power is therefore equal to $\frac{2}{s^*}$. Since $\phi(x)$ is monotonically decreasing on $0 \le x < \infty$, we conclude that $f(s, t)$ is monotonically increasing on both $0 < s < \infty$ and $0 \le t < \infty$. By finite induction, we conclude that for all $s > s^*$, $t_\ell(s) > t_\ell(s^*)$ and $t_\ell(s)$ will converge to ∞.

Lemma 2: $t_\ell(s)$ will converge to ∞ iff

$$t < f(s, t) \tag{13}$$

for all $t \in \mathbb{R}^+$.

Proof: Let us first assume $t < f(s,t)\ \forall\ t \in \mathbb{R}^+$. Since $t_{\ell+1} > t_\ell$, t_ℓ will converge to $t_\infty \in (0, \infty]$. However, t_∞ cannot be finite, because if so then $t_\infty = f(s, t_\infty)$.

Now we assume $t' = f(s, t')$ for some $t' \in \mathbb{R}^+$. $t' \ne 0$ because $f(s, 0) > 0$. Since f is continuous, there exists $t_0 > 0$ such that $t < f(s,t)$ for $\forall\ t \in (0, t_0)$. Now we choose the smallest $t' \in \mathbb{R}^+$ such that $t' = f(s, t')$ and $t < f(s,t)$ for $\forall\ t \in (0, t')$. Since $0 < f(s, 0) < f(s, t') = t'$ and $f(s,t) < f(s,t') = t'$ for $\forall\ t \in (0, t')$, we conclude that $t_\ell < t'$ for $\forall\ \ell \ge 0$. □

As an alternative expression to (11), for $0 < s < \infty$ and $0 < r \le 1$, we define $h_i(s, r)$ and $h(s, r)$ as

$$\begin{aligned} h_i(s,r) &= \phi\left(s + (i-1)\sum_{j=2}^{d_r} \rho_j \phi^{-1}(1 - (1-r)^{j-1})\right) \\ h(s,r) &= \sum_{i=2}^{d_l} \lambda_i h_i(s,r). \end{aligned} \tag{14}$$

Now, (12) becomes equivalent to

$$r_\ell = h(s, r_{\ell-1}) \tag{15}$$

where $s = m_{u_0}$. The initial value r_0 is $\phi(s)$. It is easy to see that $t_\ell \to \infty$ iff $r_\ell \to 0$. Thus, the convergence condition $r_\ell(s) \to 0$ is satisfied iff

$$r > h(s, r) \qquad \forall\ r \in (0, \phi(s)). \tag{16}$$

This will be useful in optimizing $\lambda(x)$, since (14) is linear in the λ_i's.

Let us take an example to demonstrate how well message densities are approximated by Gaussian mixtures. The following rate-$1/2$ code (17) was designed using density evolution and has the threshold 0.9669 [10]. The threshold calculated by the Gaussian approximation is 0.9473.

$$\begin{aligned} \lambda(x) &= 0.23403x + 0.21242x^2 + 0.14690x^5 + 0.10284x^6 \\ &\quad + 0.30381x^{19} \\ \rho(x) &= 0.71875x^7 + 0.28125x^8. \end{aligned} \tag{17}$$

Fig. 4 shows two message densities from variable to check nodes when density evolution and the Gaussian approximation are used for the code in (17). The noise power was set to be 0.006 dB below the threshold in each case. Surprisingly, they are very close. Fig. 5 shows corresponding densities from check to variable nodes. In this case, the exact density has a spike at 0, which cannot be modeled well by a Gaussian approximation. Even though the two densities look very different, the fact that the two densities in the other direction are very close suggests that the approximation is working well. In fact, even though the exact density in Fig. 5 is

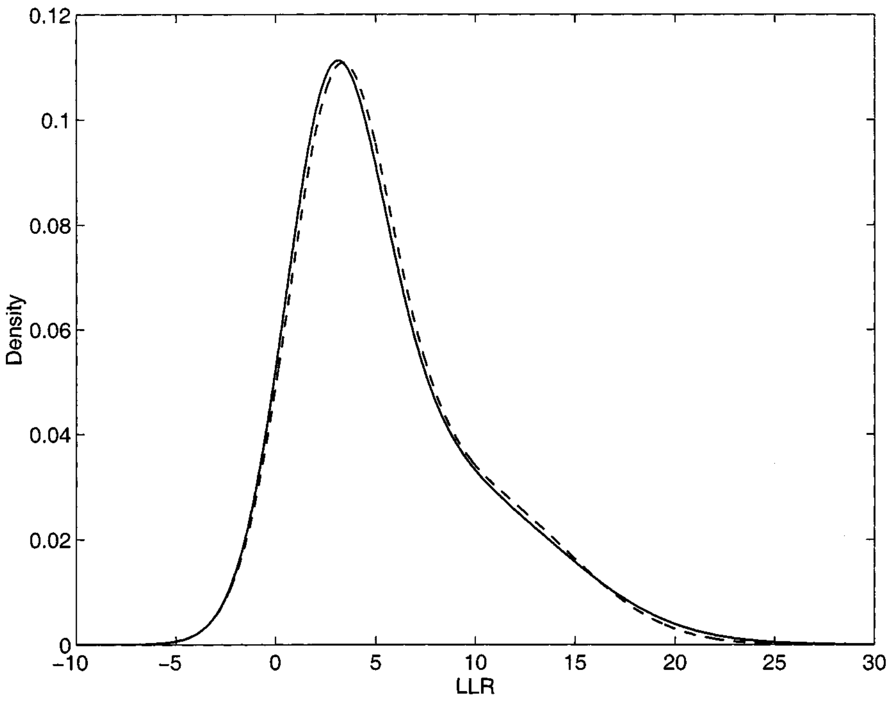

Fig. 4. Approximate (- -) and exact (—) densities at the check node input.

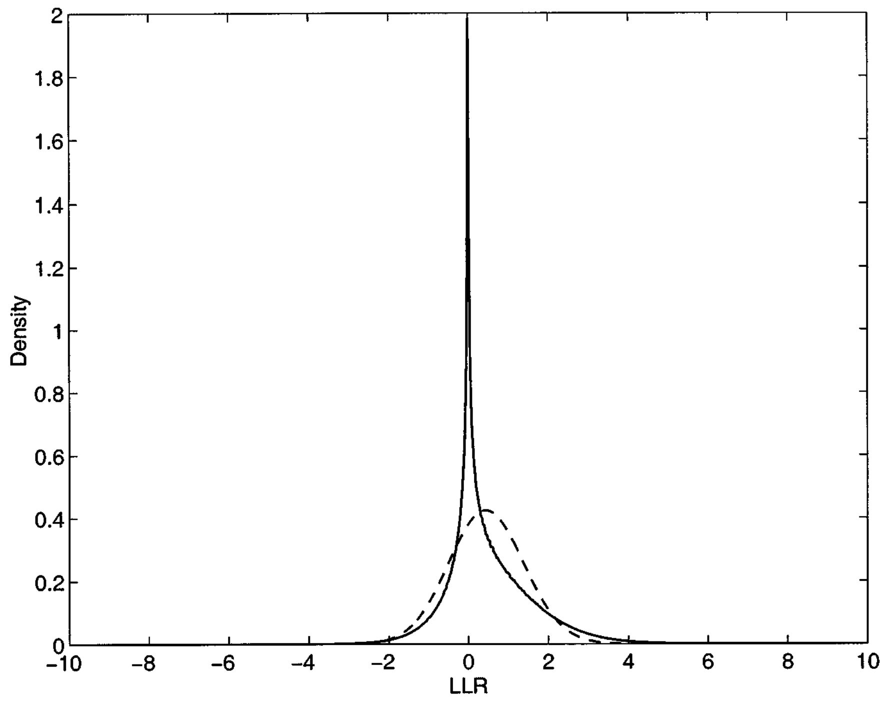

Fig. 5. Approximate (- -) and exact (—) densities at the variable node input.

very different from a Gaussian, the output of the variable node will be Gaussian-like again because many i.i.d. inputs are added at the variable node. Furthermore, when the probability of error is small enough, the exact density in Fig. 5 becomes Gaussian-like, as observed in [13]. We note that such a mismatch between the exact and approximate densities in Fig. 5 still exists in regular cases, too, because the shape of the exact density in Fig. 5 is due to the nature of check node computations (the $\tanh$ rule), not because $\rho(x)$ is irregular.

IV. Stability

In this section, we find the condition for the convergence of t_ℓ to ∞ in (12) as ℓ tends to infinity when t_0 is large enough, also known as the stability condition [13]. We also derive the rate of convergence of the probability of error using this result.

We first prove the following lemma, which gives the behavior of (11) when t is large (equivalently, small probability of error).

Lemma 3: As $t \to \infty$, $\Delta t = f(s, t) - t$ becomes

$$\Delta t = s + (i_1 - 2)t - 4\log\lambda_{i_1} - 4\sum_{j=2}^{d_r}\rho_j \log(j-1) + O(t^{-1}) \tag{18}$$

where λ_{i_1} is the first nonzero λ_i.

Proof: When t is large in (11), $\sum_{i=2}^{d_l}\lambda_i\phi(s+(i-1)t)$ can be simplified to

$$\sum_{i=2}^{d_l}\lambda_i\phi(s+(i-1)t) = \lambda_{i_1}\phi(s+(i_1-1)t) + O(\lambda_{i_2}\phi(s+(i_2-1)t))$$

where λ_{i_1} and λ_{i_2} are the first and the second nonzero λ_i's, respectively. If $\lambda_k = 1$ for some $k \geq 2$, then we define $i_2 = k+1$. Using $(1+x)^n = 1 + nx + O(x^2)$, we get

$$f_j(s, t) = \phi^{-1}((j-1)\lambda_{i_1}\phi(s+(i_1-1)t) \cdot [1 + O(\phi(s+(i_1-1)t))]) \tag{19}$$

where we used $\phi(s+(i_2-1)t) \ll \phi(s+(i_1-1)t)$. By taking $\phi(\cdot)$ on both sides of (19), we get

$$\begin{aligned}\phi(f_j(s,t)) &= (j-1)\lambda_{i_1}\phi(s+(i_1-1)t) \cdot [1+O(\phi(s+(i_1-1)t))] \\ &= (1+O(t^{-1}))\frac{1}{\sqrt{s+(i_1-1)t}} \cdot e^{-(s+(i_1-1)t-4\log(j-1)\lambda_{i_1})/4}\end{aligned}$$

where we used $\phi(s+(i_1-1)t) \ll t^{-1}$ and Lemma 1. By using Lemma 1 again, we get

$$\phi(f_j(s,t)) = \phi\left(s+(i_1-1)t - 4\log(j-1)\lambda_{i_1} + O(t^{-1})\right).$$

Finally, by taking $\phi^{-1}(\cdot)$ and by using (11), we get the result. □

If $i_1 > 2$, then t_ℓ always converges to infinity regardless of s and $\rho(x)$ as ℓ tends to infinity if t_0 is large enough. Otherwise, Δt becomes a constant as t becomes large. For this case, using $s = 2/\sigma_n^2$, we get

$$\Delta t = 4\log\frac{\lambda_2^*}{\lambda_2} + O(t^{-1})$$

where

$$\lambda_2^* = e^{1/2\sigma_n^2}\Big/\prod_{j=2}^{d_r}(j-1)^{\rho_j}.$$

The following theorem summarizes these two results.

Theorem 1: If $\lambda_2 < \lambda_2^*$, then t_ℓ will converge to infinity as ℓ tends to infinity if t_0 is large enough. If $\lambda_2 > \lambda_2^*$, then t_ℓ cannot converge to infinity regardless of its initial value. Note that this condition is similar to the stability condition

$$\lambda_2 < e^{1/2\sigma_n^2}/\sum_{j=2}^{d_r}\rho_j(j-1)$$

of [13], except that $\sum_{j=2}^{d_r}\rho_j(j-1)$ is replaced by $\prod_{j=2}^{d_r}(j-1)^{\rho_j}$. Using Jensen's inequality, we conclude that

$$\prod_{j=2}^{d_r}(j-1)^{\rho_j} \leq \sum_{j=2}^{d_r}\rho_j(j-1)$$

for all $\rho(x)$, with equality iff $\rho_k = 1$ for some $k \geq 2$. Therefore, the approximate stability condition is looser than the original, which implies that stable degree distributions under the Gaussian approximation may not be stable under density evolution. However, stability under density evolution guarantees stability under the Gaussian approximation. In any case, these two are the same if $\rho_k = 1$ for some $k \geq 2$, and are very close if $\rho(x)$ is concentrated at two consecutive degrees, which is the case for most good degree distributions found so far.

If $0 < \lambda_2 < \lambda_2^*$, then for large t_ℓ we can express t_ℓ as

$$t_\ell \approx c + 4\ell\log\frac{\lambda_2^*}{\lambda_2} \tag{20}$$

where c is a constant. The probability of error P_ℓ at the ℓth iteration is given by

$$P_\ell = \sum_{i=2}^{d_l}\lambda_i' Q\left(\sqrt{\frac{s+it_\ell}{2}}\right) \tag{21}$$

where

$$\lambda_i' = \frac{\lambda_i/i}{\sum\limits_{j=2}^{d_l}\lambda_j/j}$$

is the fraction of degree-i variable nodes. When t_ℓ is large, we get the following approximation for the probability of error.

Lemma 4: When t_ℓ is large, P_ℓ is approximated by

$$P_\ell \approx \begin{cases}\frac{a}{\sqrt{b+\ell}}\left(\frac{\lambda_2}{\lambda_2^*}\right)^{2\ell}, & \text{if } 0 < \lambda_2 < \lambda_2^* \\ \frac{d}{(i_1-1)^{\ell/2}}e^{-f(i_1-1)^\ell}, & \text{if } \lambda_2 = 0\end{cases}$$

where a, b, d, f are positive constants that depend on s and the degree distributions.

Proof: When t_ℓ is large, the first nonzero term in (21) becomes dominant. When $0 < \lambda_2 < \lambda_2^*$, P_ℓ becomes

$$P_\ell \approx \lambda_2' Q\left(\sqrt{\frac{s+2t_\ell}{2}}\right) \approx \frac{a}{\sqrt{b+\ell}}\left(\frac{\lambda_2}{\lambda_2^*}\right)^{2\ell}$$

where a and b are constants that depend on the degree distributions and s, and we have used

$$Q\left(\sqrt{\frac{x}{2}}\right) = (1+O(x^{-1}))\frac{1}{\sqrt{\pi x}}e^{-x/4}$$

and (20). If $\lambda_2 = 0$, then t_ℓ becomes approximately $d'(i_1-1)^\ell$, where d' is a positive constant that depends on the degree distributions and s. Using this and the above approximation for the Q-function, the result follows. □

This implies that when $0 < \lambda_2 < \lambda_2^*$ and P_ℓ is small, the error probability P_ℓ will decrease by a factor of $(\lambda_2/\lambda_2^*)^2$ at each iteration, which matches empirical observations using density evolution very well.

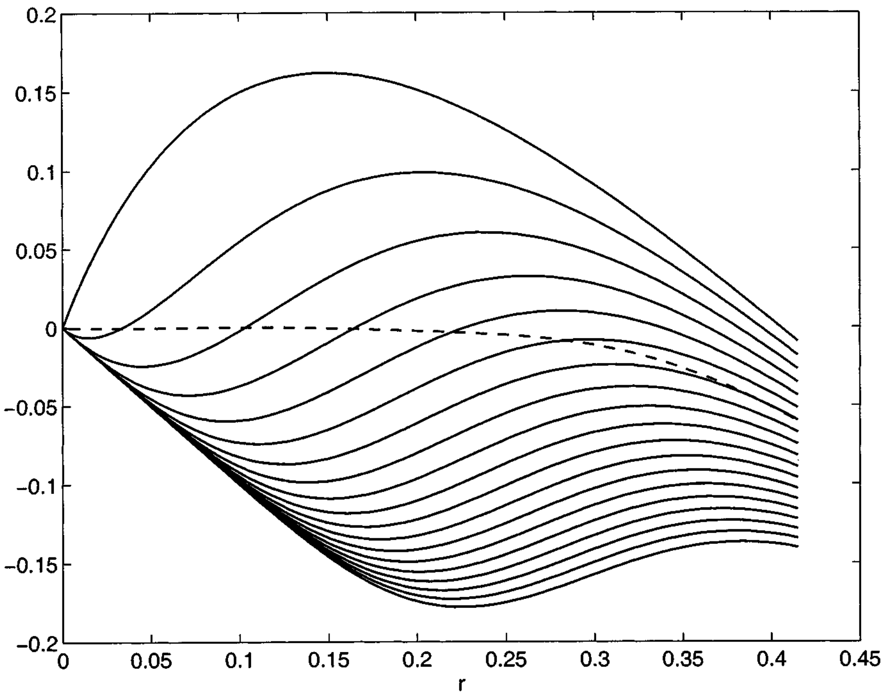

Fig. 6. $\{h_i(s, r) - r\}$ for $i = 2, \ldots, 20$ (top to bottom) and $\{h(s, r) - r\}$ (- -).

V. Optimization of Degree Distributions

Assuming that $\lambda(x)$ and σ_n are given, we can optimize $\rho(x)$ by maximizing the rate of the code. The constraints are 1) normalization: $\rho(1) = 1$; and 2) the inequality constraint (13). Since this is a linear program, it can be solved efficiently. Alternatively, however, we can maximize (18) ignoring the $O(t^{-1})$ term. As shown in the following theorem, the $\rho(x)$ optimized in this way has the form $\rho(x) = \rho x^{k-1} + (1-\rho)x^k$ for some $k \geq 2$ and $0 < \rho \leq 1$.

Theorem 2: A concentrated degree distribution

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

for some $k \geq 2$ and $0 < \rho \leq 1$, minimizes $\sum_{j=2}^{d_r} \rho_j \log(j-1)$ subject to $\rho(1) = 1$ and $\int_0^1 \rho(x)\,dx = c$, where c is a constant.

Proof: The dual linear program to the given problem is

$$\begin{aligned} \max \quad & y_1 + cy_2 \\ \text{subject to} \quad & y_1 + \frac{y_2}{j} \leq \log(j-1), \qquad 2 \leq j \leq d_r. \end{aligned}$$

Suppose we choose an integer k such that

$$\frac{1}{k+1} < c \leq \frac{1}{k}.$$

We claim that the following are feasible solutions to the primal and dual problems of equal cost, which are optimal solutions to both primal and dual problems by the weak duality theorem [29]:

$$\begin{aligned} \rho_k &= k(k+1)c - k \\ \rho_{k+1} &= 1 - \rho_k \\ y_1 &= (k+1)\log k - k\log(k-1) \\ y_2 &= k(k+1)\log\frac{k-1}{k}. \end{aligned}$$

To check whether the constraints of the dual problem are satisfied, we first observe that $y_1 + \frac{y_2}{k} = \log(k-1)$, and since

$$\begin{aligned} &\left(\log j - \frac{y_2}{j+1}\right) - \left(\log(j-1) - \frac{y_2}{j}\right) \\ &= \frac{1}{j(j+1)}\left[k(k+1)\log\frac{k-1}{k} - j(j+1)\log\frac{j-1}{j}\right] \\ &\begin{cases} < 0, & \text{if } j < k \\ = 0, & \text{if } j = k \\ > 0, & \text{if } j > k \end{cases} \end{aligned}$$

we conclude that $y_1 + \frac{y_2}{j} \leq \log(j-1)$, $j \geq 2$, by finite induction. □

Note that this solution also maximizes (18) without the $O(t^{-1})$ term, subject to the rate constraint. Experiments show that this concentrated form of $\rho(x)$ is not much different from the optimized degree distributions using linear programming, and that there is no noticeable performance degradation for the cases we have examined so far. Thus, we will use only concentrated $\rho(x)$'s.

The optimization of $\lambda(x)$ can be done similarly to the optimization of $\rho(x)$, i.e., assuming $\rho(x)$ and σ_n are given, we optimize $\lambda(x)$ by maximizing the rate

$$r = 1 - \frac{\int_0^1 \rho(x)\,dx}{\int_0^1 \lambda(x)\,dx}.$$

The constraints are 1) normalization: $\lambda(1) = 1$; and 2) the inequality constraint (16). This can be also done using linear programming.

Fig. 6 shows $\{h_i(s, r) - r\}$ for $i = 2, \ldots, 20$ using the degree distribution $\rho(x)$ in (17) where the noise level is equal to the Gaussian approximation threshold of the code in (17). In this case, where $\rho(x)$ and the noise power are given, the optimization of $\lambda(x)$ may be performed by maximizing the rate of the

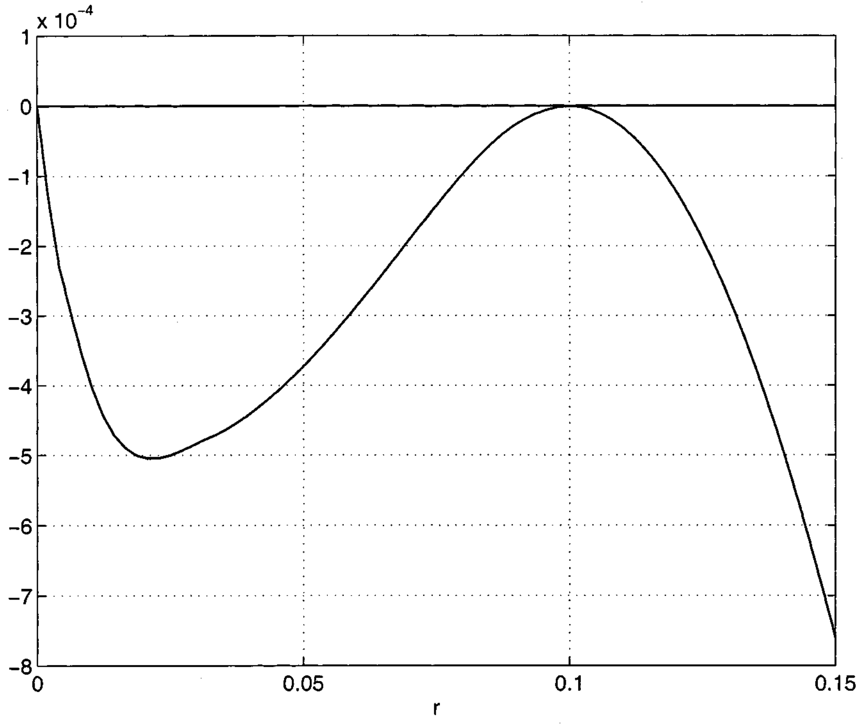

Fig. 7. $\{h(s, r) - r\}$ as a function of r magnified.

code subject to the normalization constraint $\lambda(1) = 1$ and the inequality constraints $\lambda_i \geq 0, 2 \leq i \leq d_l$, and

$$\sum_{i=2}^{d_l} \lambda_i (h_i(s, r) - r) < 0, \qquad r \in (0, \phi(s)).$$

Figs. 6 and 7 show how each curve in $\{h_i(s, r) - r\}$ for $i = 2, \ldots, 20$, is combined to produce the curve $\{h(s, r) - r\}$, where the degree distribution $\lambda(x)$ in (17) is used. This curve is always below 0 and barely touches 0 at one point. If it hits 0, then the point r^* where the curve hits 0 becomes a fixed point, and the error probability cannot be decreased further. Note that, since the slope of the curve of $\{h_2(s, r) - r\}$ near $r = 0$ is different from those for $i > 2$, the stability condition in Theorem 1 is described by only λ_2.

Although we have shown that concentrated $\rho(x)$ are good enough, we may still want to visualize how the optimization of $\rho(x)$ is done. Fig. 8 shows $\{f_i(s, t) - t\}$ for $i = 2, \ldots, 10$ using $\rho(x)$ in (17) where the noise level is equal to the Gaussian approximation threshold of the code in (17). In this case, where $\lambda(x)$ and the noise power are given, the optimization of $\rho(x)$ may be performed by maximizing the rate of the code subject to the normalization constraint $\rho(1) = 1$ and the inequality constraints $\rho_i \geq 0, 2 \leq i \leq d_r$, and

$$\sum_{i=2}^{d_r} \rho_i (f_i(s, t) - t) > 0, \qquad t \in (0, \infty).$$

Figs. 8 and 9 show how two curves corresponding to $i = 8$ and 9 in $\{f_i(s, t) - t\}$ are combined to produce the curve $\{f(s, t) - t\}$, where $\rho(x)$ in (17) is used. This curve is always above 0 and barely touches 0 at one point. If it hits 0, then the point t^* where the curve hits 0 becomes a fixed point, and the error probability cannot be decreased further.

Figs. 10 and 11 show thresholds in $\mathrm{SNR}_{\mathrm{norm}}$ for some codes of rates 0.1 to 1 optimized using both density evolution and the Gaussian approximation when the maximum variable degree d_l is 10 or 20, respectively. They show that the difference between the exact and approximate thresholds is small (less than 0.3 dB) for codes designed using the Gaussian approximation. It also shows that the optimization based on the Gaussian approximation is good especially when d_l is low and when the rate is high. For example, for rates higher than 0.5, the difference is less than 0.02 dB for $d_l = 10$ and less than 0.1 dB for $d_l = 20$.

VI. Fixed Points

We have seen that the Gaussian approximation is useful in calculating thresholds and optimizing degree distributions. However, it is not clear how accurate the approximation is by just looking at a single number, namely, a threshold. In this section, we show how accurately the Gaussian approximation can predict the locations of potential fixed points of density evolution, which could show more aspects of the approximation than thresholds.

First observe that in Figs. 7 and 9 there are slow and fast phases, where the width of the gap between the curve and the horizontal axis determines the iteration speed. This explains why there are slow and fast phases in sum-product decoding, as noted in [13].

Fig. 12 shows the decrease in the probability of error as a function of the probability of error for the $d_l = 20$ code in (17) using density evolution and the Gaussian approximation, where the noise power is set to be 0.006 dB below the threshold of each case. It is interesting to observe that there are three estimated

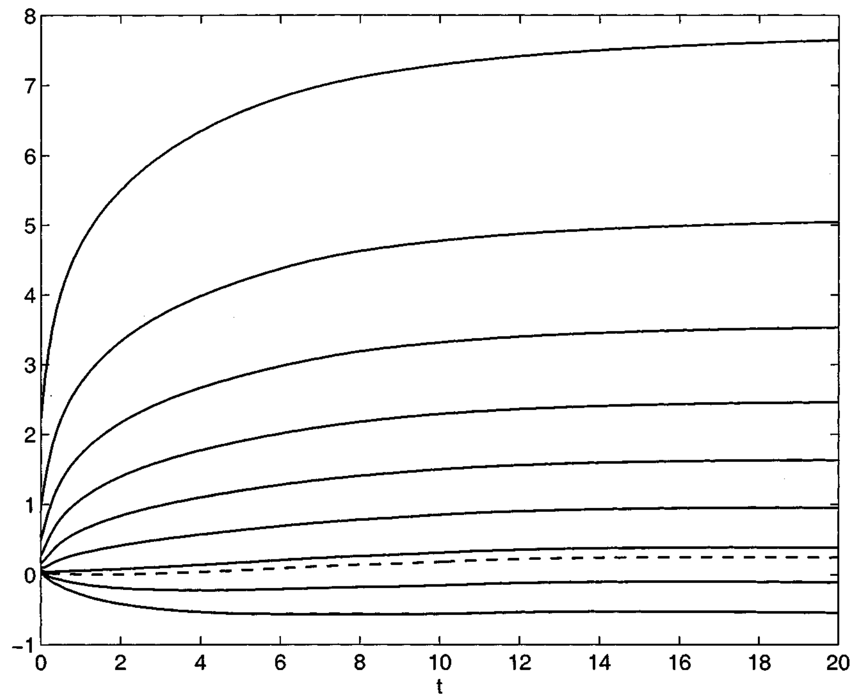

Fig. 8. $\{f_i(s, t) - t\}$ for $i = 2, \ldots, 10$ (top to bottom) and $\{f(s, t) - t\}$ (- -).

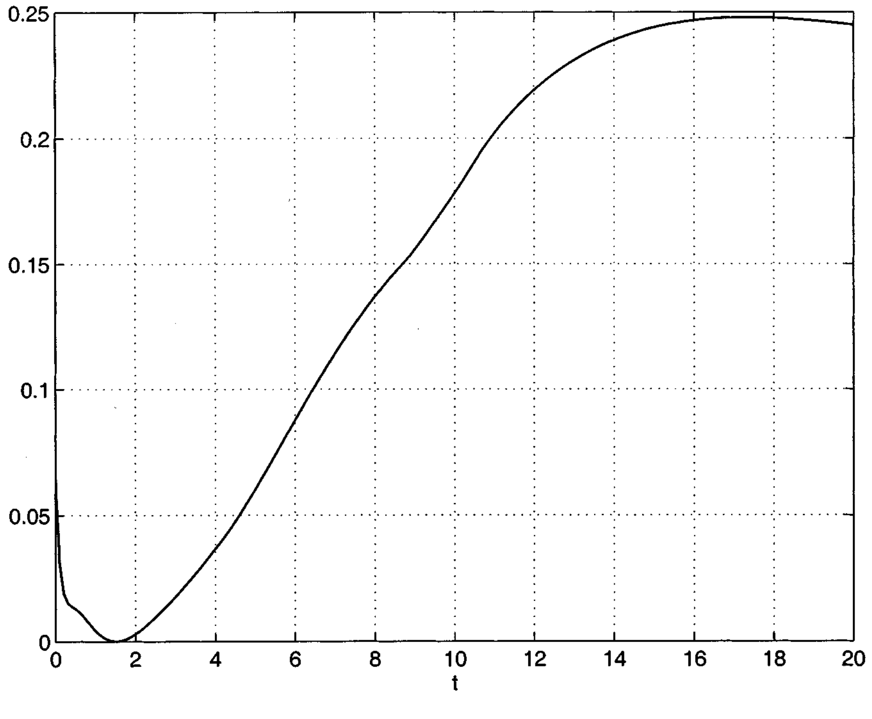

Fig. 9. $\{f(s, t) - t\}$ as a function of t magnified.

fixed points under density evolution (including zero probability of error) but only two under the Gaussian approximation. As we increase the noise power, 2×10^{-2} (probability of error) becomes a fixed point under both the Gaussian approximation and density evolution. As the noise power is increased further, 9×10^{-2} becomes another fixed point under density evolution.

Fig. 13 shows the decrease in the probability of error as a function of the probability of error for the following $d_l = 7$ code (rate $= 1/2$) (22) using density evolution and the Gaussian approximation, where the noise power is set to be 0.006 dB below the threshold of each case.

$$\begin{aligned}\lambda(x) &= 0.30780x + 0.27287x^2 + 0.41933x^6\\ \rho(x) &= 0.4x^5 + 0.6x^6.\end{aligned} \tag{22}$$

In this case, the number of potential fixed points is estimated correctly using the Gaussian approximation, and the two curves

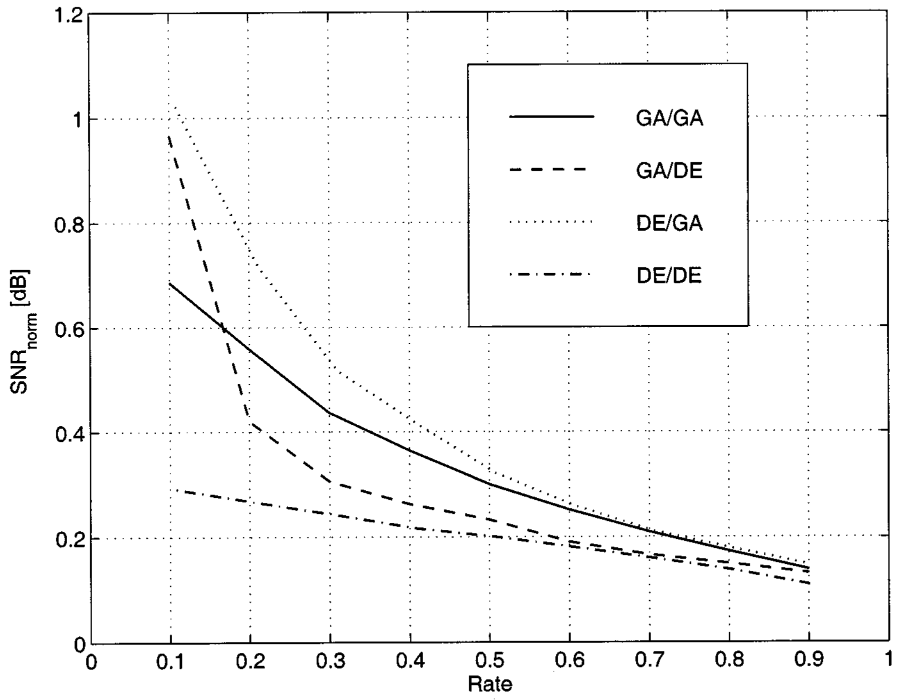

Fig. 10. Performance of various codes, designed/evaluated using GA/GA, GA/DE, DE/GA, or DE/DE when $d_l = 10$, where GA denotes the Gaussian approximation and DE denotes density evolution.

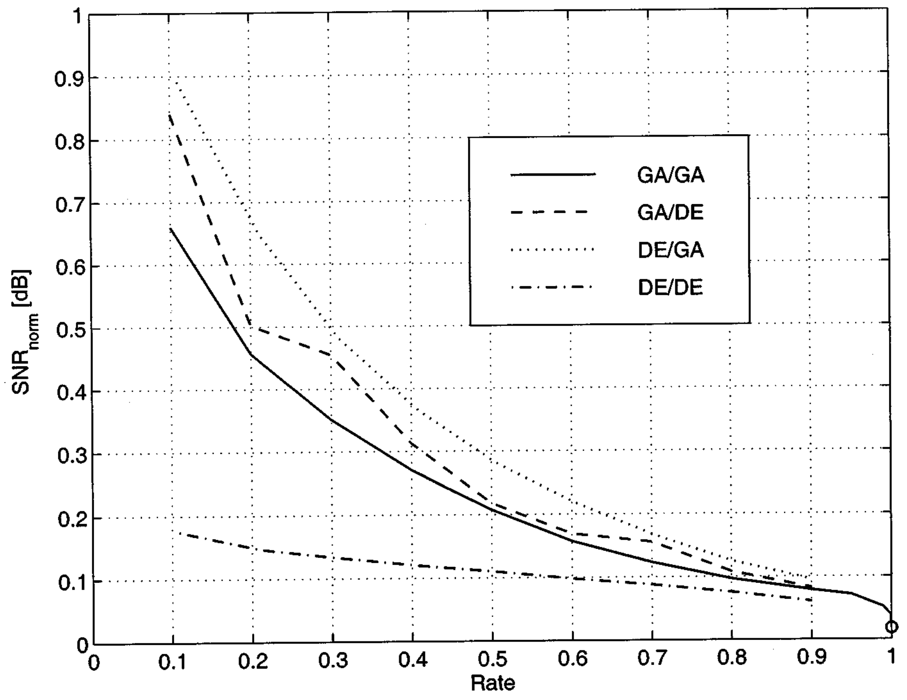

Fig. 11. Performance of various codes, designed/evaluated using GA/GA, GA/DE, DE/GA, or DE/DE when $d_l = 20$.

are quite close. In this case, the Gaussian approximation not only calculates thresholds accurately, but also tracks the exact probability of error reasonably well.

We note that, from empirical observations, as the maximum variable degree increases, the number of fixed points often tends to increase. Since the Gaussian approximation is not good at estimating the number of fixed points exactly when the maximum variable degree is large, it becomes less useful in this case. However, when the maximum variable degree is small, the Gaussian approximation works reasonably well. This seems to contradict our previous observations for regular codes, where the accuracy of the Gaussian approximation for $(j, j+1)$ codes improves as

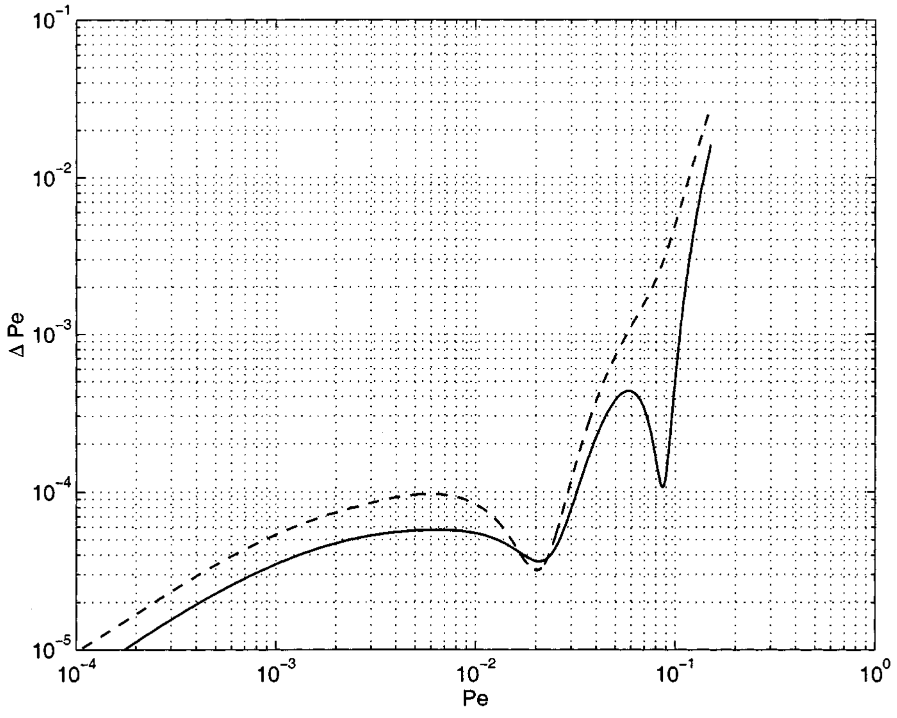

Fig. 12. The probability of error versus decrease in the probability of error of the $d_l = 20$ code in (17) using density evolution (—) and the Gaussian approximation (- -).

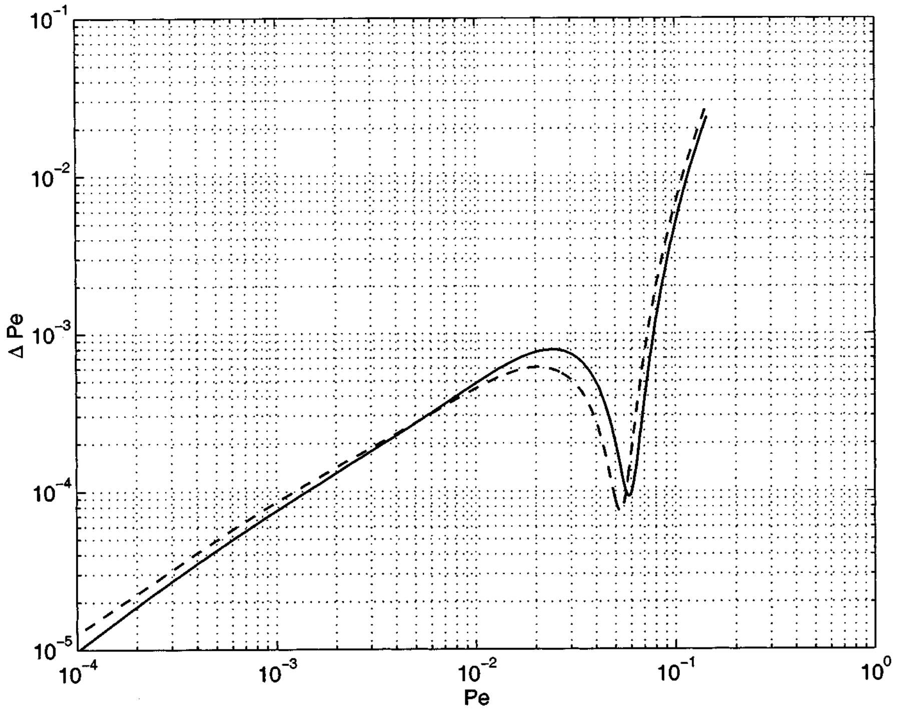

Fig. 13. The probability of error versus decrease in the probability of error for the $d_l = 7$ code in (22) using density evolution (—) and the Gaussian approximation (- -).

j increases. This seems to suggest that, as the maximum variable degree of the code increases, the Gaussian approximation becomes less accurate because the irregularity of the code increases, even though the fraction of high-degree variable nodes increases.

VII. Conclusion

For LDPC codes under sum-product decoding, the message distributions are well approximated by Gaussians or Gaussian mixtures when memoryless binary-input continuous-output

AWGN channels are used. Using the Gaussian approximation, we have simplified density evolution to a one-dimensional problem, and found simple and analyzable expressions for describing approximate density evolution. Because of the huge reduction in the dimension of the problem without much sacrifice in accuracy, we can find thresholds faster and optimize degree distributions faster and easier. It also enables us to analyze the behavior of density evolution especially near the zero probability of error, which makes it possible to find the approximate rate of convergence for the probability of error when it is near zero. The Gaussian approximation is also a good tool to visualize how the mean of densities are updated under density evolution, which also explains why there are fast and slow phases in density evolution. It also helps us get some hints on how to optimize degree distributions. Graphically, we have demonstrated how the optimization of degree distributions can be visualized.

Online demonstration of optimization of degree distributions using the Gaussian approximation and more results are available at http://truth.mit.edu/~sychung.

Acknowledgment

The authors would like to express their appreciation to Prof. G. D. Forney, Jr. for his many helpful comments on this paper. We would also like to thank anonymous reviewers and Igal Sason for helpful comments and suggestions.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.
[2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[3] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.
[4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low-density parity-check codes," *Electron. Lett.*, vol. 32, pp. 1645–1646, Aug. 1996.
[5] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.
[6] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. 30th Annu. ACM Symp. Theory of Computing*, 1998, pp. 249–258.
[7] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.
[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.
[9] 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. Lett.*, to be published.
[10] S.-Y. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, MA, 2000. Available at http://truth.mit.edu/~sychung.
[11] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. 29th Annu. ACM Symp. Theory of Computing*, 1997, pp. 150–159.
[12] 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. Information Theory*, Sorrento, Italy, June 2000, p. 318.
[13] T. J. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.
[14] S. ten Brink, "Iterative decoding trajectories of parallel concatenated codes," in *3rd IEEE ITG Conf. Source and Channel Coding*, Munich, Germany, Jan. 2000.
[15] ——, "Rate one-half code for approaching the Shannon limit by 0.1 dB," *Electron. Lett.*, vol. 36, pp. 1293–1294, July 2000.
[16] H. El Gamal and A. R. Hammons Jr., "Analyzing the turbo decoder using the Gaussian approximation," in *Proc. Int. Symp. Information Theory*, Sorrento, Italy, June 2000, p. 319.
[17] ——, "Analyzing the turbo decoder using the Gaussian approximation," *IEEE Trans. Inform. Theory*, vol. 47, pp. 000–000, Feb. 2001.
[18] D. Divsalar, S. Dolinar, and F. Pollara, "Low complexity turbo-like codes," in *Proc. Int. Symp. Turbo Codes*, Brest, France, 2000.
[19] B. Frey, "Turbo factor analysis," *Proc. Adv. Inform. Processing Syst.*, vol. 12, Dec. 1999, submitted for publication.
[20] P. Rusmevichientong and B. Van Roy, "An analysis of turbo decoding with Gaussian priors," *Proc. Adv. Neural Inform. Processing Syst.*, vol. 12, Dec. 1999.
[21] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Proc. Adv. Neural Inform. Processing Syst.*, vol. 12, Dec. 1999.
[22] G. D. Forney Jr., "Codes on graphs: Normal realizations," *IEEE Trans. Inform. Theory*, vol. 47, pp. 520–548, Feb. 2001.
[23] C. R. Hartmann and L. D. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 514–517, Sept. 1976.
[24] G. Battail, M. C. Decouvelaere, and P. Godlewski, "Replication decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 332–345, May 1979.
[25] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.
[26] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Univ. Linköping, Linköping, Sweden, 1996.
[27] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*. New York: Wiley, 1965.
[28] I. S. Gradshtein and I. M. Ryzhik, *Table of Integrals, Series, and Products*. New York: Academic, 1965.
[29] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, MA: Athena Scientific, 1997.
[30] H. Jin, A. Khandekar, and R. J. McEliece, "Irregular repeat-accumulate codes," in *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sept. 4, 2000, pp. 1–8.
[31] P. Rusmevichientong and B. Van Roy, "An analysis of belief propagation on the turbo decoding graph with Gaussian densities," *IEEE Trans. Inform. Theory*, vol. 47, pp. 745–765, Feb. 2001.