Chapter 13

Capacity-approaching codes

We have previously discussed codes on graphs and the sum-product decoding algorithm in general terms. In this chapter we will give a brief overview of some particular classes of codes that can approach the Shannon limit quite closely: low-density parity-check (LDPC) codes, turbo codes, and repeat-accumulate (RA) codes. We will analyze long LDPC codes on the binary erasure channel (BEC), where exact results can be obtained. We will also sketch how to analyze any of these codes on symmetric binary-input channels.

13.1 LDPC codes

The oldest of these classes of codes is LDPC codes, invented by Gallager in his doctoral thesis (1961). These codes were introduced long before their time ("a bit of 21st-century coding that happened to fall in the 20th century"), and were almost forgotten until the mid-1990s, after the introduction of turbo codes, when they were independently rediscovered. Because of their simple structure, they have been the focus of much analysis. They have also proved to be capable of approaching the Shannon limit more closely than any other class of codes.

An LDPC code is based on the parity-check representation of a binary linear (n, k) block code $\mathcal{C}$; *i.e.,* $\mathcal{C}$ is the set of all binary n-tuples that satisfy the $n - k$ parity-check equations

$$\mathbf{x}H^T = \mathbf{0},$$

where H is a given $(n-k) \times n$ parity-check matrix. The basic idea of an LDPC code is that n should be large and H should be sparse; *i.e.,* the density of ones in H should be of the order of a small constant times n rather than n^2. As we will see, this sparseness makes it feasible to decode $\mathcal{C}$ by iterative sum-product decoding in linear time. Moreover, H should be pseudo-random, so that $\mathcal{C}$ will be a "random-like" code.

In Chapter 11, we saw that a parity-check representation of an (n, k) linear code leads to a Tanner graph with n variable nodes, $n - k$ constraint (zero-sum) nodes, and no state nodes. The number of edges is equal to the number of ones in the parity-check matrix H. This was illustrated by the Tanner graph of a parity-check representation for an $(8, 4, 4)$ code, shown again here as Figure 1.

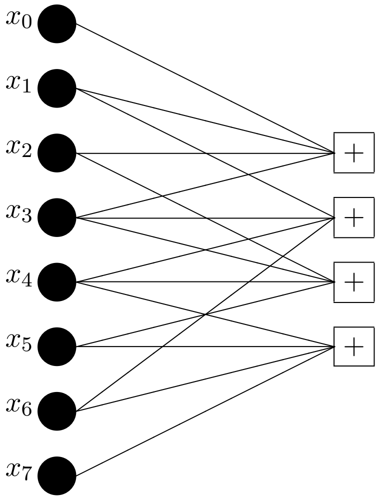

Figure 1. Tanner graph of parity-check representation for $(8, 4, 4)$ code.

Gallager's original LDPC codes were *regular*, meaning that every variable is involved in the same number d_λ of constraints, whereas every constraint checks the same number d_ρ of variables, where d_ρ and d_λ are small integers. The number of edges is thus $nd_\lambda = (n-k)d_\rho$, so the nominal rate $R = k/n$ of the code satisfies[1]

$$1 - R = \frac{n-k}{n} = \frac{d_\lambda}{d_\rho}.$$

Subject to this constraint, the connections between the nd_λ variable "sockets" and the $(n-k)d_\rho$ constraint sockets are made pseudo-randomly.

For example, the normal graph of a Gallager code with $d_\lambda = 3, d_\rho = 6$ and $R = \frac{1}{2}$ is shown in Figure 2. The large box labelled Π represents a pseudo-random permutation ("interleaver").

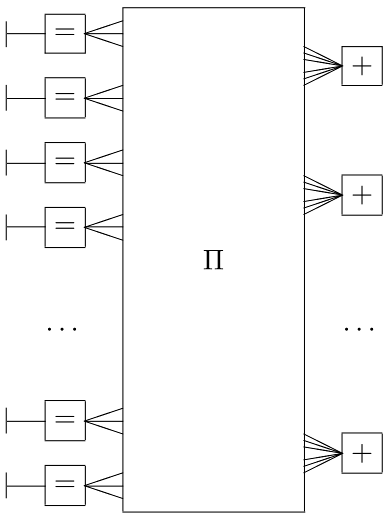

Figure 2. Normal graph of a regular $d_\lambda = 3, d_\rho = 6$ LDPC code.

[1]The actual rate of the code will be greater than R if the checks are not linearly independent. However, this fine point may be safely ignored.

Decoding an LDPC code is done by an iterative version of the sum-product algorithm, with a schedule that alternates between left nodes (repetition constraints) and right nodes (zero-sum constraints). The sparseness of the graph tends to insure that its girth (minimum cycle length) is reasonably large, so that the graph is locally tree-like. In decoding, this implies that the independence assumption holds for quite a few iterations. However, typically the number of iterations is large, so the independence assumption eventually becomes invalid.

The minimum distance of an LDPC code is typically large, so that actual decoding errors are hardly ever made. Rather, the typical decoding failure mode is a failure of the decoding algorithm to converge, which is of course a detectable failure.

The main improvement in recent years to Gallager's LDPC codes has been the use of irregular codes— *i.e.,* LDPC codes in which the left (variable) vertices and right (check) vertices have arbitrary degree distributions. The behavior of the decoding algorithm can be analyzed rather precisely using a technique called "density evolution," and the degree distributions can consequently be optimized, as we will discuss later in this chapter.

13.2 Turbo codes

The invention of turbo codes by Berrou *et al.* (1993) ignited great excitement about capacity-approaching codes. Initially, turbo codes were the class of capacity-approaching codes that were most widely used in practice, although LDPC codes may now be superseding turbo codes. Turbo codes achieve fairly low error rates within 1–2 dB of the Shannon limit at moderate block lengths ($n = 10^3$ to 10^4).

The original turbo codes of Berrou *et al.* are still some of the best that are known. Figure 3 shows a typical Berrou-type turbo code. An information bit sequence is encoded twice: first by an ordinary rate-$\frac{1}{2}$ systematic recursive (with feedback) convolutional encoder, and then, after a large pseudo-random permutation Π, by a second such encoder. The information sequence and the two parity sequences are transmitted, so the overall rate is $R = \frac{1}{3}$.

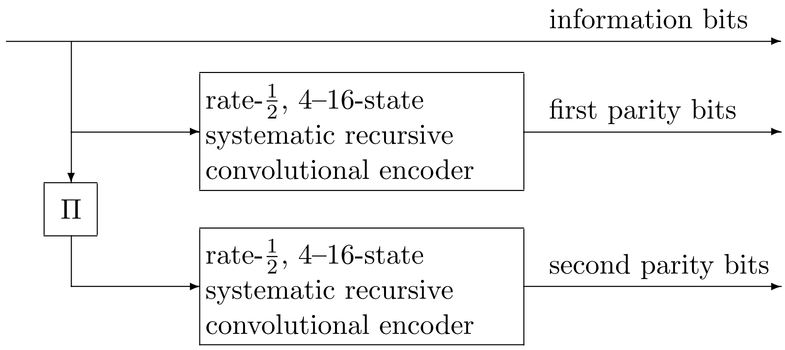

Figure 3. Rate-$\frac{1}{3}$ Berrou-type turbo code.

The two convolutional encoders are not very complicated, typically 4–16 states, and are often chosen to be identical. The convolutional codes are usually terminated to a finite block length. In practice the rate is often increased by puncturing.

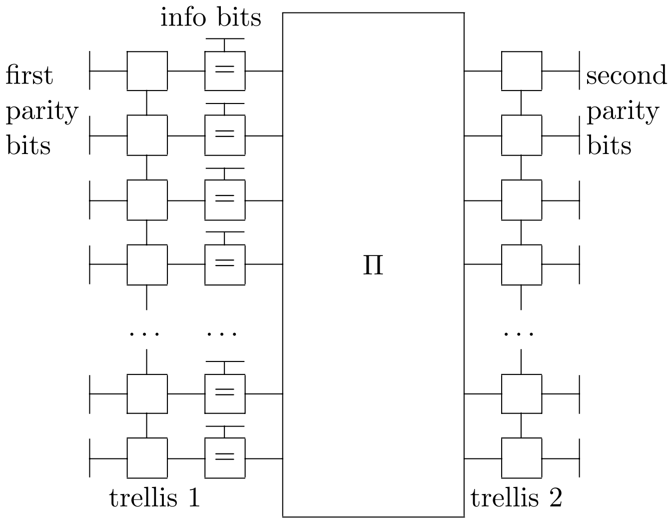

Figure 4. Normal graph of a Berrou-type turbo code.

Figure 4 is a normal graph of such a code. On both sides of the permutation Π are normal graphs representing the trellises of the two constituent convolutional codes, as in Figure 4(b) of Chapter 11. The information and parity bits are shown separately. The information bit sequences for the two trellises are identical, apart from the permutation Π.

Decoding a turbo code is done by an iterative version of the sum-product algorithm, with a schedule that alternates between the left trellis and the right trellis. On each trellis the sum-product algorithm reduces to the BCJR (APP) algorithm, so the decoder can efficiently decode the entire trellis before exchanging the resulting "extrinsic information" with the other trellis.

Again, the large permutation Π, combined with the recursive property of the encoders, tends to ensure that the girth of the graph is reasonably large, so the independence assumption holds for quite a few iterations. In turbo decoding the number of iterations is typically only 10–20, since much computation can be done along a trellis in each iteration.

The minimum distance of a Berrou-type turbo code is typically not very large. Although at low SNRs the decoding error probability tends to drop off rapidly above a threshold SNR (the "waterfall region") down to $10^{-4}, 10^{-5}$ or lower, for higher SNRs the error probability falls off more slowly due to low-weight error events (the "noise floor" region), and the decoder actually makes undetectable errors.

For applications in which these effects are undesirable, a different arrangement of two constituent codes is used, namely one after the other as in classical concatenated coding. Such codes are called "serial concatenated codes," whereas the Berrou-type codes are called "parallel concatenated codes." Serial concatenated codes usually still have an error floor, but typically at a considerably lower error rate than parallel concatenated codes. On the other hand, their threshold in the waterfall region tends to be worse than that of parallel concatenated codes. The "repeat-accumulate" codes of the next section are simple serial concatenated codes.

Analysis of turbo codes tends to be more *ad hoc* than that of LDPC codes. However, good *ad hoc* techniques for estimating decoder performance are now known (*e.g.,* the "extrinsic information transfer (EXIT) chart;" see below), which allow optimization of the component codes.

13.3 Repeat-accumulate codes

Repeat-accumulate (RA) codes were introduced by Divsalar, McEliece *et al.* (1998) as extremely simple “turbo-like” codes for which there was some hope of proving theorems. Surprisingly, even such simple codes proved to work quite well, within about 1.5 dB of the Shannon limit— *i.e.,* better than the best schemes known prior to turbo codes.

RA codes are a very simple class of serial concatenated codes. The outer code is a simple $(n, 1, n)$ repetition code, which simply repeats the information bits n times. The resulting sequence is then permuted by a large pseudo-random permutation Π. The inner code is a rate-1 2-state convolutional code with generator $g(D) = 1/(1 + D)$; *i.e.,* the input/output equation is $y_k = x_k + y_{k-1}$, so the output bit is simply the “accumulation” of all previous input bits (mod 2). The complete RA encoder is shown in Figure 5.

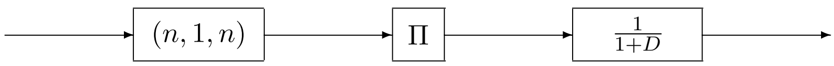

Figure 5. Rate-$\frac{1}{n}$ RA encoder.

The normal graph of a rate-$\frac{1}{3}$ RA code is shown in Figure 6. Since the original information bits are not transmitted, they are regarded as hidden state variables, repeated three times. On the right side, the states of the 2-state trellis are the output bits y_k, and the trellis constraints are represented explicitly by zero-sum nodes that enforce the constraints $y_k + x_k + y_{k-1} = 0$.

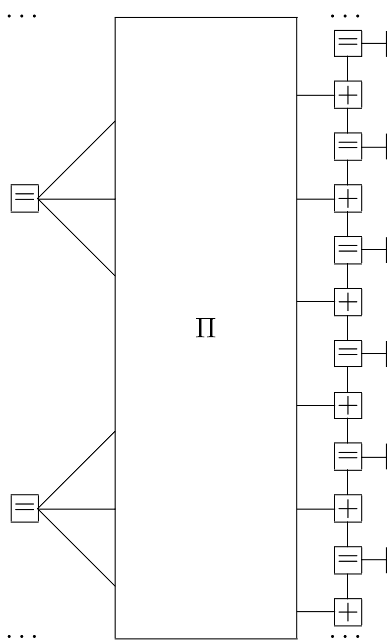

Figure 6. Normal graph of rate-$\frac{1}{3}$ RA code.

Decoding an RA code is again done by an iterative version of the sum-product algorithm, with a schedule that alternates between the left constraints and the right constraints, which in this case form a 2-state trellis. For the latter, the sum-product algorithm again reduces to the BCJR (APP) algorithm, so the decoder can efficiently decode the entire trellis in one iteration. A left-side iteration does not accomplish as much, but on the other hand it is extremely simple. As with LDPC codes, performance may be improved by making the left degrees irregular.

13.4 Analysis of LDPC codes on the binary erasure channel

In this section we will analyze the performance of iterative decoding of long LDPC codes on a binary erasure channel. This is one of the few scenarios in which exact analysis is possible. However, the results are qualitatively (and to a considerable extent quantitatively) indicative of what happens in more general scenarios.

13.4.1 The binary erasure channel

The binary erasure channel (BEC) models a memoryless channel with two inputs $\{0, 1\}$ and three outputs, $\{0, 1, ?\}$, where "?" is an "erasure symbol." The probability that any transmitted bit will be received correctly is $1 - p$, that it will be erased is p, and that it will be received incorrectly is zero. These transition probabilities are summarized in Figure 7 below.

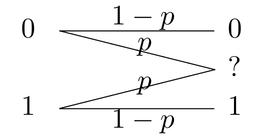

Figure 7. Transition probabilities of the binary erasure channel.

The binary erasure channel is an exceptional channel, in that a received symbol either specifies the transmitted symbol completely, or else gives no information about it. There are few physical examples of such binary-input channels. However, a Q-ary erasure channel (QEC) is a good model of packet transmission on the Internet, where (because of internal parity checks on packets) a packet is either received perfectly or not at all.

If a code sequence $\mathbf{c}$ from a binary code $\mathcal{C}$ is transmitted over a BEC, then the received sequence $\mathbf{r}$ will agree with $\mathbf{c}$ in all unerased symbols. If there is no other code sequence $\mathbf{c}' \in \mathcal{C}$ that agrees with $\mathbf{r}$ in all unerased symbols, then $\mathbf{c}$ is the only possible transmitted sequence, so a maximum-likelihood (ML) decoder can decide that $\mathbf{c}$ was sent with complete confidence. On the other hand, if there is another code sequence $\mathbf{c}' \in \mathcal{C}$ that agrees with $\mathbf{r}$ in all unerased symbols, then there is no way to decide between $\mathbf{c}$ and $\mathbf{c}'$, so a detectable decoding failure must occur. (We consider a random choice between $\mathbf{c}$ and $\mathbf{c}'$ to be a decoding failure.)

The channel capacity of a BEC with erasure probability p is $1 - p$, the fraction of unerased symbols, as would be expected intuitively. If feedback is available, then the channel capacity may be achieved simply by requesting retransmission of each erased symbol (the method used on the Internet Q-ary erasure channel).

Even without feedback, if we choose the 2^{nR} code sequences in a block code $\mathcal{C}$ of length n and rate R independently at random with each bit having probability $\frac{1}{2}$ of being 0 or 1, then as $n \to \infty$ the probability of a code sequence $\mathbf{c}' \in \mathcal{C}$ agreeing with the transmitted sequence $\mathbf{c}$ in all $\approx n(1 - p)$ unerased symbols is about $2^{-n(1-p)}$, so by the union bound estimate the probability of decoding failure is about

$$\Pr(E) \approx 2^{nR} \cdot 2^{-n(1-p)},$$

which decreases exponentially with n as long as $R < 1 - p$. Thus capacity can be approached arbitrarily closely without feedback. On the other hand, if $R > 1-p$, then with high probability there will be only $\approx n(1 - p) < nR$ unerased symbols, which can distinguish between at most $2^{n(1-p)} < 2^{nR}$ code sequences, so decoding must fail with high probability.

13.4.2 Iterative decoding of LDPC codes on the BEC

On the binary erasure channel, the sum-product algorithm is greatly simplified, because at any time every variable corresponding to every edge in the code graph is either known perfectly (unerased) or not known at all (erased). Iterative decoding using the sum-product algorithm therefore reduces simply to the propagation of unerased variables through the code graph.

There are only two types of nodes in a normal graph of an LDPC code (*e.g.,* Figure 2): repetition nodes and zero-sum nodes. If all variables are either correct or erased, then the sum-product update rule for a repetition node reduces simply to:

> If any incident variable is unerased, then all other incident variables may be set equal to that variable, with complete confidence; otherwise, all incident variables remain erased.

For a zero-sum node, the sum-product update rule reduces to:

> If all but one incident variable is unerased, then the remaining incident variable may be set equal to the mod-2 sum of those inputs, with complete confidence; otherwise, variable assignments remain unchanged.

Since all unerased variables are correct, there is no chance that these rules could produce a variable assignment that conflicts with another assignment.

Exercise 1. Using a graph of the $(8, 4, 4)$ code like that of Figure 1 for iterative decoding, decode the received sequence $(1, 0, 0, ?, 0, ?, ?, ?)$. Then try to decode the received sequence $(1, 1, 1, 1, ?, ?, ?, ?)$. Why does decoding fail in the latter case? Give both a local answer (based on the graph) and a global answer (based on the code). For the received sequence $(1, 1, 1, 1, ?, ?, ?, 0)$, show that iterative decoding fails but that global (*i.e.,* ML) decoding succeeds. □

13.4.3 Performance of large random LDPC codes

We now analyze the performance of iterative decoding for asymptotically large random LDPC codes on the BEC. Our method is a special case of a general method called *density evolution*, and is illustrated by a special case of the *EXtrinsic Information Transfer (EXIT) chart* technique.

We first analyze a random ensemble of regular (d_ρ, d_λ) LDPC codes of length n and rate $R = 1 - d_\lambda/d_\rho$. In this case all left nodes are repetition nodes of degree $d_\lambda + 1$, with one external (input) incident variable and d_λ internal (state) incident variables, and all right nodes are zero-sum nodes of degree d_ρ, with all incident variables being internal (state) variables. The random element is the permutation Π, which is chosen equiprobably from the set of all $(nd_\lambda)!$ permutations of $nd_\lambda = (n - k)d_\rho$ variables. We let $n \to \infty$ with (d_ρ, d_λ) fixed.

We use the standard sum-product algorithm, which alternates between sum-product updates of all left nodes and all right nodes. We track the progress of the algorithm by the expected fraction q of internal (state) variables that are still erased at each iteration. We will assume that at each node, the incident variables are independent of each other and of everything else; as $n \to \infty$, this "locally tree-like" assumption is justified by the large random interleaver.

At a repetition node, if the current probability of internal variable erasure is $q_{\rm in}$, then the probability that a given internal incident variable will be erased as a result of its sum-product update is the probability that both the external incident variable and all $d_\lambda - 1$ other internal incident variables are erased, namely $q_{\rm out} = p(q_{\rm in})^{d_\lambda - 1}$.

On the other hand, at a zero-sum node, the probability that a given incident variable will not be erased is the probability that all $d_\rho - 1$ other internal incident variables are not erased, namely $(1 - q_{\text{in}})^{d_\rho - 1}$, so the probability that it *is* erased is $q_{\text{out}} = 1 - (1 - q_{\text{in}})^{d_\rho - 1}$.

These two functions are plotted in the "EXIT chart" of Figure 8 in the following manner. The two variable axes are denoted by $q_{r\to\ell}$ and $q_{\ell\to r}$, where the subscripts denote respectively left-going and right-going erasure probabilities. The range of both axes is from 1 (the initial value) to 0 (hopefully the final value). The two curves represent the sum-product update relationships derived above:

$$q_{\ell\to r} = p(q_{r\to\ell})^{d_\lambda - 1}; \qquad q_{r\to\ell} = 1 - (1 - q_{\ell\to r})^{d_\rho - 1}.$$

These curves are plotted for a regular $(d_\lambda = 3, d_\rho = 6)$ $(R = \frac{1}{2})$ LDPC code on a BEC with $p = 0.4$.

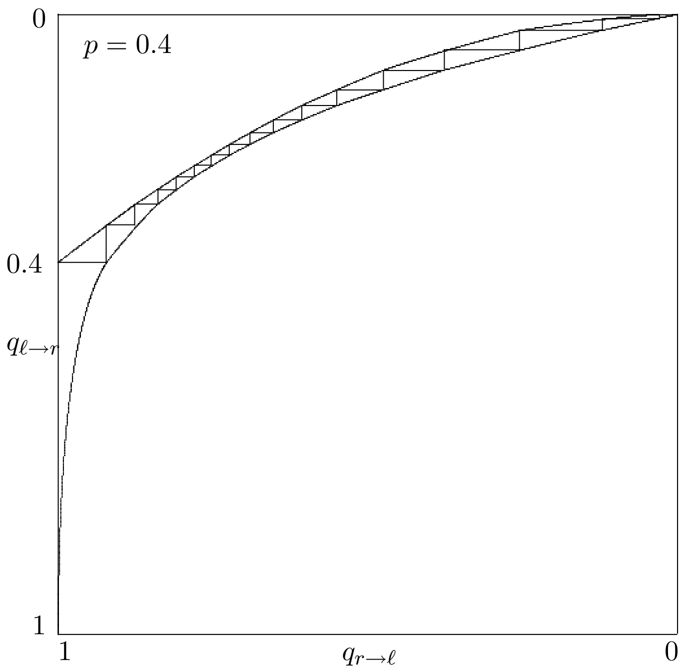

Figure 8. EXIT chart for iterative decoding of a regular $(d_\lambda = 3, d_\rho = 6)$ LDPC code on a BEC with $p = 0.4$.

A "simulation" of iterative decoding may then be performed as follows (also plotted in Figure 8). Initially, the left-going erasure probability is $q_{r\to\ell} = 1$. After a sum-product update in the left (repetition) nodes, the right-going erasure probability becomes $q_{\ell\to r} = p = 0.4$. After a sum-product update in the right (zero-sum) nodes, the left-going erasure probability becomes $q_{r\to\ell} = 1 - (0.6)^5 = 0.922$. Continuing, the erasure probability evolves as follows:

$q_{r\to\ell}$: 1 ↘ 0.922 ↘ 0.875 ↘ 0.839 ↘ 0.809 ↘ 0.780 ↘ 0.752 ↘ 0.723 ↘ 0.690 ↘ 0.653 ↘ 0.607 ↘ 0.550 ↘ 0.475 ↘ 0.377 ↘ …

$q_{\ell\to r}$: 0.400 ↗ 0.340 ↗ 0.306 ↗ 0.282 ↗ 0.262 ↗ 0.244 ↗ 0.227 ↗ 0.209 ↗ 0.191 ↗ 0.170 ↗ 0.147 ↗ 0.121 ↗ 0.090 ↗ 0.057 ↗

We see that iterative decoding must eventually drive the erasure probabilities to the top right corner of Figure 8, $q_{\ell\to r} = q_{r\to\ell} = 0$, because the two curves do not cross. It takes quite a few iterations, about 15 in this case, to get through the narrow "tunnel" where the two curves approach each other closely. However, once past the "tunnel," convergence is rapid. Indeed, when both erasure probabilities are small, the result of one complete (left and right) iteration is

$$q_{r\to\ell}^{\text{new}} \approx 5q_{\ell\to r} = 5p(q_{r\to\ell}^{\text{old}})^2.$$

Thus both probabilities decrease rapidly (doubly exponentially) with the number of iterations.

The iterative decoder will fail if and only if the two curves touch; *i.e.,* if and only if there exists a pair $(q_{\ell\to r}, q_{r\to\ell})$ such that $q_{\ell\to r} = p(q_{r\to\ell})^{d_\lambda-1}$ and $q_{r\to\ell} = 1-(1-q_{\ell\to r})^{d_\rho-1}$, or equivalently the single-iteration update equation

$$q_{r\to\ell}^{\text{new}} = 1-(1-p(q_{r\to\ell}^{\text{old}})^{d_\lambda-1})^{d_\rho-1}$$

has a fixed point with $q_{r\to\ell}^{\text{new}} = q_{r\to\ell}^{\text{old}}$. For example, if $p = 0.45$ and $d_\lambda = 3, d_\rho = 6$, then this equation has a (first) fixed point at about $(q_{\ell\to r} \approx 0.35, q_{r\to\ell} \approx 0.89)$, as shown in Figure 9.

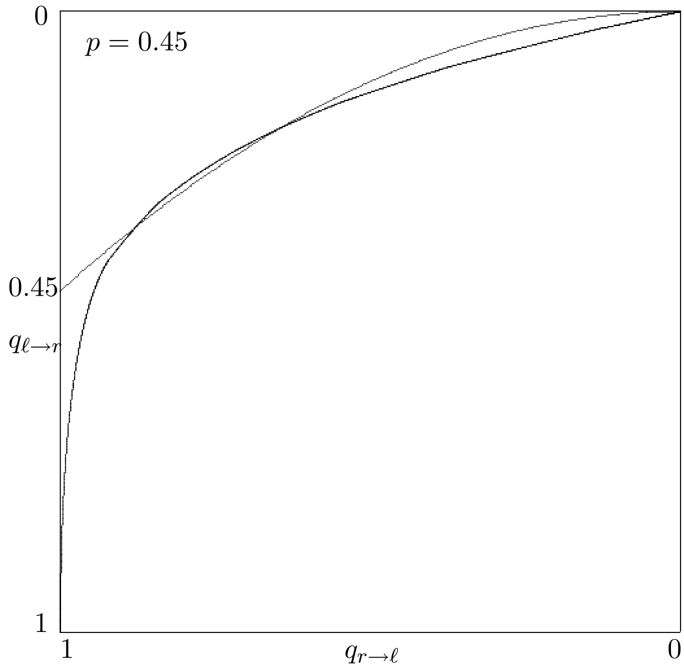

Figure 9. EXIT chart for iterative decoding of a regular $(d_\lambda = 3, d_\rho = 6)$ LDPC code on a BEC with $p = 0.45$.

Exercise 2. Perform a simulation of iterative decoding of a regular $(d_\lambda = 3, d_\rho = 6)$ LDPC code on a BEC with $p = 0.45$ (*i.e.,* on Figure 9), and show how decoding gets stuck at the first fixed point $(q_{\ell\to r} \approx 0.35, q_{r\to\ell} \approx 0.89)$. About how many iterations does it take to get stuck? By simulation of iterative decoding, compute the coordinates of the fixed point to six significant digits. □

We conclude that iterative decoding of a large regular $(d_\lambda = 3, d_\rho = 6)$ LDPC code, which has nominal rate $R = \frac{1}{2}$, will succeed on a BEC when the channel erasure probability is less than some threshold p^*, where $0.4 < p^* < 0.45$. The threshold p^* is the smallest p such that the equation $x = 1-(1-px^2)^5$ has a solution in the interval $0 < x < 1$.

Exercise 3. By analysis or simulation, show that $p^* = 0.429....$

13.4.4 Analysis of irregular LDPC codes

Now let us apply a similar analysis to large *irregular* LDPC codes, where the left nodes and/or right nodes do not necessarily all have the same degree. We characterize ensembles of such codes by the following parameters.

The number of external variables and left (repetition) nodes is the code length n; again we let $n \to \infty$. The number of internal variables (edges) will be denoted by E, which we will allow to grow linearly with n. The number of right (zero-sum) nodes will be $n - k = n(1 - R)$, yielding a nominal code rate of R.

A left node will be said to have degree d if it has d incident internal edges (*i.e.,* the external variable is not counted in its degree). The number of left nodes of degree d will be denoted by L_d. Thus $n = \sum_d L_d$. Similarly, the number of right nodes of degree d will be denoted by R_d, and $n(1 - R) = \sum_d R_d$. Thus the nominal rate R is given by

$$R = 1 - \frac{\sum_d R_d}{\sum_d L_d}.$$

An edge will be said to have left degree d if it is incident on a left node of degree d. The number of edges of left degree d will be denoted by ℓ_d; thus $\ell_d = dL_d$. Similarly, the number of edges of right degree d is $r_d = dR_d$. The total number of edges is thus

$$E = \sum_d \ell_d = \sum_d r_d.$$

It is helpful to define generating functions of these degree distributions as follows:

$$\begin{aligned} L(x) &= \sum_d L_d x^d; \\ R(x) &= \sum_d R_d x^d; \\ \ell(x) &= \sum_d \ell_d x^{d-1}; \\ r(x) &= \sum_d r_d x^{d-1}. \end{aligned}$$

Note that $L(1) = n$, $R(1) = n(1-R)$, and $\ell(1) = r(1) = E$. Also, note that $\ell(x)$ is the derivative of $L(x)$,

$$L'(x) = \sum_d L_d d x^{d-1} = \sum_d \ell_d x^{d-1} = \ell(x),$$

and similarly $R'(x) = r(x)$. Conversely, we have the integrals

$$\begin{aligned} L(x) &= \sum_d (\ell_d/d) x^d = \int_0^x \ell(y)dy; \\ R(x) &= \sum_d (r_d/d) x^d = \int_0^x r(y)dy. \end{aligned}$$

Finally, we have

$$R = 1 - \frac{R(1)}{L(1)} = 1 - \frac{\int_0^1 r(x)dx}{\int_0^1 \ell(x)dx}.$$

In the literature, it is common to normalize all of these generating functions by the total number of edges E; *i.e.,* we define $\lambda(x) = \ell(x)/E = \ell(x)/\ell(1)$, $\rho(x) = r(x)/E = r(x)/r(1)$, $\Lambda(x) = L(x)/E = \int_0^x \lambda(y)dy$, and $\mathrm{P}(x) = R(x)/E = \int_0^x \rho(y)dy$. In these terms, we have

$$R = 1 - \frac{\mathrm{P}(1)}{\Lambda(1)} = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}.$$

The *average left degree* is defined as $\overline{d}_\lambda = E/n$, and the *average right degree* as $\overline{d}_\rho = E/n(1-R)$; therefore $1/\overline{d}_\lambda = \Lambda(1)$, $1/\overline{d}_\rho = \mathrm{P}(1)$, and

$$R = 1 - \frac{\overline{d}_\lambda}{\overline{d}_\rho}.$$

The analysis of iterative decoding of irregular LDPC codes may be carried out nicely in terms of these generating functions. At a left node, if the current left-going probability of variable erasure is $q_{r\to\ell}$, then the probability that a given internal variable of left degree d will be erased as a result of a sum-product update is $p(q_{r\to\ell})^{d-1}$. The expected fraction of erased right-going variables is thus

$$q_{\ell\to r} = p \sum_d \lambda_d (q_{r\to\ell})^{d-1},$$

where $\lambda_d = \ell_d/E$ is the fraction of edges of left degree d. Thus

$$q_{\ell\to r} = p\lambda(q_{r\to\ell}),$$

where $\lambda(x) = \sum_d \lambda_d x^{d-1}$. Similarly, the expected fraction of erased left-going variables after a sum-product update at a right node is

$$q_{r\to\ell} = \sum_d \rho_d \left(1 - (1 - q_{\ell\to r})^{d-1}\right) = 1 - \rho(1 - q_{\ell\to r}),$$

where $\rho_d = r_d/E$ is the fraction of edges of right degree d, and $\rho(x) = \sum_d \rho_d x^{d-1}$.

These equations generalize the equations $q_{\ell\to r} = p(q_{r\to\ell})^{\lambda-1}$ and $q_{r\to\ell} = 1 - (1 - q_{\ell\to r})^{\rho-1}$ for the regular case. Again, these two curves may be plotted in an EXIT chart, and may be used for an exact calculation of the evolution of the erasure probabilities $q_{\ell\to r}$ and $q_{r\to\ell}$ under iterative decoding. And again, iterative decoding will be successful if and only if the two curves do not cross. The fixed-point equation now becomes

$$x = 1 - \rho(1 - p\lambda(x)).$$

Design of a capacity-approaching LDPC code therefore becomes a matter of choosing the left and right degree distributions $\lambda(x)$ and $\rho(x)$ so that the two curves come as close to each other as possible, without touching.

13.4.5 Area theorem

The following lemma and theorem show that in order to approach the capacity $C = 1 - p$ of the BEC arbitrarily closely, the two EXIT curves must approach each other arbitrarily closely; moreover, if the rate R exceeds C, then the two curves must cross.

Lemma 13.1 (Area theorem) *The area under the curve $q_{\ell \to r} = p\lambda(q_{r \to \ell})$ is $p/\overline{d}_\lambda$, while the area under the curve $q_{r \to \ell} = 1 - \rho(1 - q_{\ell \to r})$ is $1 - 1/\overline{d}_\rho$.*

Proof.

$$\begin{aligned} \int_0^1 p\lambda(x)dx &= p\Lambda(1) = \frac{p}{\overline{d}_\lambda}; \\ \int_0^1 (1 - \rho(1-x))dx &= \int_0^1 (1 - \rho(y))dy = 1 - \mathrm{P}(1) = 1 - \frac{1}{\overline{d}_\rho}. \end{aligned}$$

Theorem 13.2 (Converse capacity theorem) *For successful iterative decoding of an irregular LDPC code on a BEC with erasure probability p, the code rate R must be less than the capacity $C = 1 - p$. As $R \to C$, the two EXIT curves must approach each other closely, but not cross.*

Proof. For successful decoding, the two EXIT curves must not intersect, which implies that the regions below the two curves must be disjoint. This implies that the sum of the areas of these regions must be less than the area of the EXIT chart, which is 1:

$$p\Lambda(1) + 1 - \mathrm{P}(1) < 1.$$

This implies that $p < \mathrm{P}(1)/\Lambda(1) = 1 - R$, or equivalently $R < 1 - p = C$. If $R \approx C$, then $p\Lambda(1) + 1 - \mathrm{P}(1) \approx 1$, which implies that the union of the two regions must nearly fill the whole EXIT chart, whereas for successful decoding the two regions must remain disjoint. □

13.4.6 Stability condition

Another necessary condition for the two EXIT curves not to cross is obtained by considering the curves near the top right point $(0, 0)$. For $q_{r \to \ell}$ small, we have the linear approximation

$$q_{\ell \to r} = p\lambda(q_{r \to \ell}) \approx p\lambda'(0)q_{r \to \ell}.$$

Similarly, for $q_{\ell \to r}$ small, we have

$$q_{r \to \ell} = 1 - \rho(1 - q_{\ell \to r}) \approx 1 - \rho(1) + \rho'(1)q_{\ell \to r} = \rho'(1)q_{\ell \to r},$$

where we use $\rho(1) = \sum_d \rho_d = 1$. After one complete iteration, we therefore have

$$q_{r \to \ell}^{\mathrm{new}} \approx p\lambda'(0)\rho'(1)q_{r \to \ell}^{\mathrm{old}},$$

The erasure probability $q_{r \to \ell}$ is thus reduced on a complete iteration if and only if

$$p\lambda'(0)\rho'(1) < 1.$$

This is known as the *stability condition* on the degree distributions $\lambda(x)$ and $\rho(x)$. Graphically, it ensures that the line $q_{\ell \to r} \approx p\lambda'(0)q_{r \to \ell}$ lies above the line $q_{r \to \ell} \approx \rho'(1)q_{\ell \to r}$ near the point $(0, 0)$, which is a necessary condition for the two curves not to cross.

Exercise 4. Show that if the minimum left degree is 3, then the stability condition necessarily holds. Argue that such a degree distribution $\lambda(x)$ cannot be capacity-approaching, however, in view of Theorem 13.2. □

Luby, Shokrollahi *et al.* have shown that for any $R < C$ it is possible to design $\lambda(x)$ and $\rho(x)$ so that the nominal code rate is R and the two curves do not cross, so that iterative decoding will be successful. Thus iterative decoding of LDPC codes solves the longstanding problem of approaching capacity arbitrarily closely on the BEC with a linear-time decoding algorithm, at least for asymptotically long codes.

13.5 LDPC code analysis on symmetric binary-input channels

In this final section, we will sketch how to analyze a long LDPC code on a general symmetric binary-input channel (SBIC). Density evolution is exact, but is difficult to compute. EXIT charts give good approximate results.

13.5.1 Symmetric binary-input channels

The binary symmetric channel (BSC) models a memoryless channel with binary inputs $\{0, 1\}$ and binary outputs $\{0, 1\}$. The probability that any transmitted bit will be received correctly is $1 - p$, and that it will be received incorrectly is p. This model is depicted in Figure 10 below.

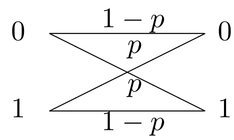

Figure 10. Transition probabilities of the binary symmetric channel.

By symmetry, the capacity of a BSC is attained when the two input symbols are equiprobable. In this case, given a received symbol y, the *a posteriori* probabilities $\{p(0 \mid y), p(1 \mid y)\}$ of the transmitted symbols are $\{1 - p, p\}$ for $y = 0$ and $\{p, 1 - p\}$ for $y = 1$. The conditional entropy $H(X \mid y)$ is thus equal to the binary entropy function $\mathcal{H}(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$, independent of y. The channel capacity is therefore equal to $C = H(X) - H(X \mid Y) = 1 - \mathcal{H}(p)$.

A general *symmetric binary-input channel* (SBIC) is a channel with binary inputs $\{0, 1\}$ that may be viewed as a mixture of binary symmetric channels, as well as possibly a binary erasure channel. In other words, the channel output alphabet may be partitioned into pairs $\{a, b\}$ such that $p(a \mid 0) = p(b \mid 1)$ and $p(a \mid 1) = p(b \mid 0)$, as well as possibly a singleton $\{?\}$ such that $p(? \mid 0) = p(? \mid 1)$.

Again, by symmetry, the capacity of a SBIC is attained when the two input symbols are equiprobable. Then the *a posteriori* probabilities (APPs) are symmetric, in the sense that for any such pair $\{a, b\}$, $\{p(0 \mid y), p(1 \mid y)\}$ equals $\{1 - p, p\}$ for $y = a$ and $\{p, 1 - p\}$ for $y = b$, where

$$p = \frac{p(a \mid 1)}{p(a \mid 0) + p(a \mid 1)} = \frac{p(b \mid 0)}{p(b \mid 0) + p(b \mid 1)}.$$

Similarly, $\{p(0 \mid ?), p(1 \mid ?)\} = \{\frac{1}{2}, \frac{1}{2}\}$.

We may characterize a symmetric binary-input channel by the probability distribution of the APP parameter $p = p(1 \mid y)$ under the conditional distribution $p(y \mid 0)$, which is the same as the distribution of $p = p(0 \mid y)$ under $p(y \mid 1)$. This probability distribution is in general continuous if the output distribution is continuous, or discrete if it is discrete.[2]

Example 1. Consider a binary-input channel with five outputs $\{y_1, y_2, y_3, y_4, y_5\}$ such that $\{p(y_j \mid 0)\} = \{p_1, p_2, p_3, p_4, p_5\}$ and $\{p(y_j \mid 1)\} = \{p_5, p_4, p_3, p_2, p_1\}$. This channel is a SBIC, because the outputs may be grouped into symmetric pairs $\{y_1, y_5\}$ and $\{y_2, y_4\}$ and a singleton $\{y_3\}$ satisfying the symmetry conditions given above. The values of the APP parameter p are then $\{\frac{p_5}{p_1+p_5}, \frac{p_4}{p_2+p_4}, \frac{1}{2}, \frac{p_2}{p_2+p_4}, \frac{p_1}{p_1+p_5}\}$; their probability distribution is $\{p_1, p_2, p_3, p_4, p_5\}$. □

Example 2. Consider a binary-input Gaussian-noise channel with inputs $\{\pm 1\}$ and Gaussian conditional probability density $p(y \mid x) = (2\pi\sigma)^{-1/2} \exp -(y-x)^2/2\sigma^2$. This channel is a SBIC, because the outputs may be grouped into symmetric pairs $\{\pm y\}$ and a singleton $\{0\}$ satisfying the symmetry conditions given above. The APP parameter is then

$$p = \frac{p(y \mid -1)}{p(y \mid 1) + p(y \mid -1)} = \frac{e^{-y/\sigma^2}}{e^{y/\sigma^2} + e^{-y/\sigma^2}},$$

whose probability density is induced by the conditional density $p(y \mid +1)$. □

Given an output y, the conditional entropy $H(X \mid y)$ is given by the binary entropy function, $H(X \mid y) = \mathcal{H}(p(0 \mid y)) = \mathcal{H}(p(1 \mid y))$. The average conditional entropy is thus

$$H(X \mid Y) = \mathsf{E}_{Y|0}[H(X \mid y)] = \int dy \; p(y \mid 0) \; \mathcal{H}(p(0 \mid y)),$$

where we use notation that is appropriate for the continuous case. The channel capacity is then $C = H(X) - H(X \mid Y) = 1 - H(X \mid Y)$ bits per symbol.

Example 1. For the five-output SBIC of Example 1,

$$H(X \mid Y) = (p_1 + p_5)\mathcal{H}\left(\frac{p_1}{p_1 + p_5}\right) + (p_2 + p_4)\mathcal{H}\left(\frac{p_2}{p_2 + p_4}\right) + p_3. \qquad \square$$

Example 3. A binary erasure channel with erasure probability p is a SBIC with probabilities $\{1-p, p, 0\}$ of the APP parameter being $\{0, \frac{1}{2}, 1\}$, and thus of $H(X \mid y)$ being $\{0, 1, 0\}$. Therefore $H(X \mid Y) = p$ and $C = 1 - p$. □

13.5.2 Sum-product decoding of an LDPC code on a SBIC

For an LDPC code on a SBIC, the sum-product iterative decoding algorithm is still quite simple, because all variables are binary, and all constraint nodes are either repetition or zero-sum nodes.

For binary variables, APP vectors are always of the form $\{1 - p, p\}$, up to scale; *i.e.,* they are specified by a single parameter. We will not worry about scale in implementing the sum-product algorithm, so in general we will simply have unnormalized APP weights $\{w_0, w_1\}$, from which can recover p using the equation $p = w_1/(w_0 + w_1)$. Some implementations use likelihood ratios $\lambda = w_0/w_1$, from which we can recover p using $p = 1/(1 + \lambda)$. Some use log likelihood ratios $\Lambda = \ln \lambda$, from which we can recover p using $p = 1/(1 + e^{\Lambda})$.

[2]Note that any outputs with the same APP parameter may be combined without loss of optimality, since the APPs form a set of sufficient statistics for estimation of the input from the output.

For a repetition node, the sum-product update rule reduces simply to the product update rule, in which the components of the incoming APP vectors are multiplied componentwise: *i.e.,*

$$w_0^{\rm out} = \prod_j w_0^{{\rm in},j}; \qquad w_1^{\rm out} = \prod_j w_1^{{\rm in},j},$$

where $\{w_{x_j}^{{\rm in},j} \mid x_j \in \{0,1\}\}$ is the jth incoming APP weight vector. Equivalently, the output likelihood ratio is the product of the incoming likelihood ratios; or, the output log likelihood ratio is the sum of the incoming log likelihood ratios.

For a zero-sum node, the sum-product update rule reduces to

$$w_0^{\rm out} = \sum_{\sum x_j = 0} \prod_j w_{x_j}^{{\rm in},j}; \qquad w_1^{\rm out} = \sum_{\sum x_j = 1} \prod_j w_{x_j}^{{\rm in},j}.$$

An efficient implementation of this rule is as follows:[3]

(a) Transform each incoming APP weight vector $\{w_0^{{\rm in},j}, w_1^{{\rm in},j}\}$ by a 2×2 Hadamard transform to $\{W_0^{{\rm in},j} = w_0^{{\rm in},j} + w_1^{{\rm in},j}, W_1^{{\rm in},j} = w_0^{{\rm in},j} - w_1^{{\rm in},j}\}$.

(b) Form the componentwise product of the transformed vectors; *i.e.,*

$$W_0^{\rm out} = \prod_j W_0^{{\rm in},j}; \qquad W_1^{\rm out} = \prod_j W_1^{{\rm in},j}.$$

(c) Transform the outgoing APP weight vector $\{W_0^{\rm out}, W_1^{\rm out}\}$ by a 2×2 Hadamard transform to $\{w_0^{\rm out} = W_0^{\rm out} + W_1^{\rm out}, w_1^{\rm out} = W_0^{\rm out} - W_1^{\rm out}\}$.

Exercise 5. (Sum-product update rule for zero-sum nodes)

(a) Prove that the above algorithm implements the sum-product update rule for a zero-sum node, up to scale. [Hint: observe that in the product $\prod_j (w_0^{{\rm in},j} - w_1^{{\rm in},j})$, the terms with positive signs sum to $w_0^{\rm out}$, whereas the terms with negative signs sum to $w_1^{\rm out}$.]

(b) Show that if we interchange $w_0^{{\rm in},j}$ and $w_1^{{\rm in},j}$ in an even number of incoming APP vectors, then the outgoing APP vector $\{w_0^{\rm out}, w_1^{\rm out}\}$ is unchanged. On the other hand, show that if we interchange $w_0^{{\rm in},j}$ and $w_1^{{\rm in},j}$ in an odd number of incoming APP vectors, then the components $w_0^{\rm out}$ and $w_1^{\rm out}$ of the outgoing APP vector are interchanged.

(c) Show that if we replace APP weight vectors $\{w_0, w_1\}$ by log likelihood ratios $\Lambda = \ln w_0/w_1$, then the zero-sum sum-product update rule reduces to the "tanh rule"

$$\Lambda^{\rm out} = \ln \left(\frac{1 + \prod_j \tanh \Lambda^{{\rm in},j}/2}{1 - \prod_j \tanh \Lambda^{{\rm in},j}/2} \right),$$

where the hyperbolic tangent is defined by $\tanh x = (e^x - e^{-x})/(e^x + e^{-x})$.

(d) Show that the "tanh rule" may alternatively be written as

$$\tanh \Lambda^{\rm out}/2 = \prod_j \tanh \Lambda^{{\rm in},j}/2.$$

[3] This implementation is an instance of a more general principle: the sum-product update rule for a constraint code $\mathcal{C}$ may be implemented by Fourier-transforming the incoming APP weight vectors, performing a sum-product update for the dual code $\mathcal{C}^\perp$, and then Fourier-transforming the resulting outgoing APP weight vectors. For a binary alphabet, the Fourier transform reduces to the 2×2 Hadamard transform.

13.5.3 Density evolution

The performance of iterative decoding for asymptotically large random LDPC codes on a general SBIC may in principle be simulated exactly by tracking the probability distribution of the APP parameter p (or an equivalent parameter). This is called *density evolution.* However, whereas on the BEC there are only two possible values of p, $\frac{1}{2}$ (complete ignorance) and 0 (complete certainty), so that we need to track only the probability q of the former, in density evolution we need to track a general probability distribution for p. In practice, this cannot be done with infinite precision, so density evolution becomes inexact to some extent.

On an SBIC, the channel symmetry leads to an important simplification of density evolution: we may always assume that the all-zero codeword was sent, which means that for each variable we need to track only the distribution of p given that the value of the variable is 0.

To justify this simplification, note that any other codeword imposes a configuration of variables on the code graph such that all local constraints are satisfied; *i.e.,* all variables incident on a repetition node are equal to 0 or to 1, while the values of the set of variables incident on a zero-sum node include an even number of 1s. Now note that at a repetition node, if we interchange $w_0^{\mathrm{in},j}$ and $w_1^{\mathrm{in},j}$ in all incoming APP vectors, then the components w_0^{out} and w_1^{out} of the outgoing APP vector are interchanged. Exercise 5(b) proves a comparable result for zero-sum nodes. So if the actual codeword is not all-zero, but the channel is symmetric, so for every symbol variable which has a value 1 the initial APP vectors are simply interchanged, then the APP vectors will evolve during sum-product decoding in precisely the same way as they would have evolved if the all-zero codeword had been sent, except that wherever the underlying variable has value 1, the APP components are interchanged.

In practice, to carry out density evolution for a given SBIC with given degree distributions $\lambda(x)$ and $\rho(x)$, the probability density is quantized to a discrete probability distribution, using as many as 12–14 bits of accuracy. The repetition and check node distribution updates are performed in a computationally efficient manner, typically using fast Fourier transforms for the former and a table-driven recursive implementation of the "tanh rule" for the latter. For a given SBIC model, the simulation is run iteratively until either the distribution of p tends to a delta function at $p = 0$ (success), or else the distribution converges to a nonzero fixed point (failure). Repeated runs allow a success threshold to be determined. Finally, the degree distributions $\lambda(x)$ and $\rho(x)$ may be optimized by using hill-climbing techniques (iterative linear programming).

For example, in his thesis (2000), Chung designed rate-$\frac{1}{2}$ codes with asymptotic thresholds within 0.0045 dB of the Shannon limit, and with performance within 0.040 dB of the Shannon limit at an error rate of 10^{-6} with a block length of $n = 10^7$; see Figure 11. The former code has left degrees $\{2, 3, 6, 7, 15, 20, 50, 70, 100, 150, 400, 900, 2000, 3000, 6000, 8000\}$, with average left degree $\overline{d}_\lambda = 9.25$, and right degrees $\{18, 19\}$, with average right degree $\overline{d}_\rho = 18.5$. The latter code has left degrees $\{2, 3, 6, 7, 18, 19, 55, 56, 200\}$, with $\overline{d}_\lambda = 6$, and all right degrees equal to 12.

In current research, more structured constructions of the parity-check matrix H (or equivalently the permutation Π) are being sought for shorter block lengths, of the order of 1000.

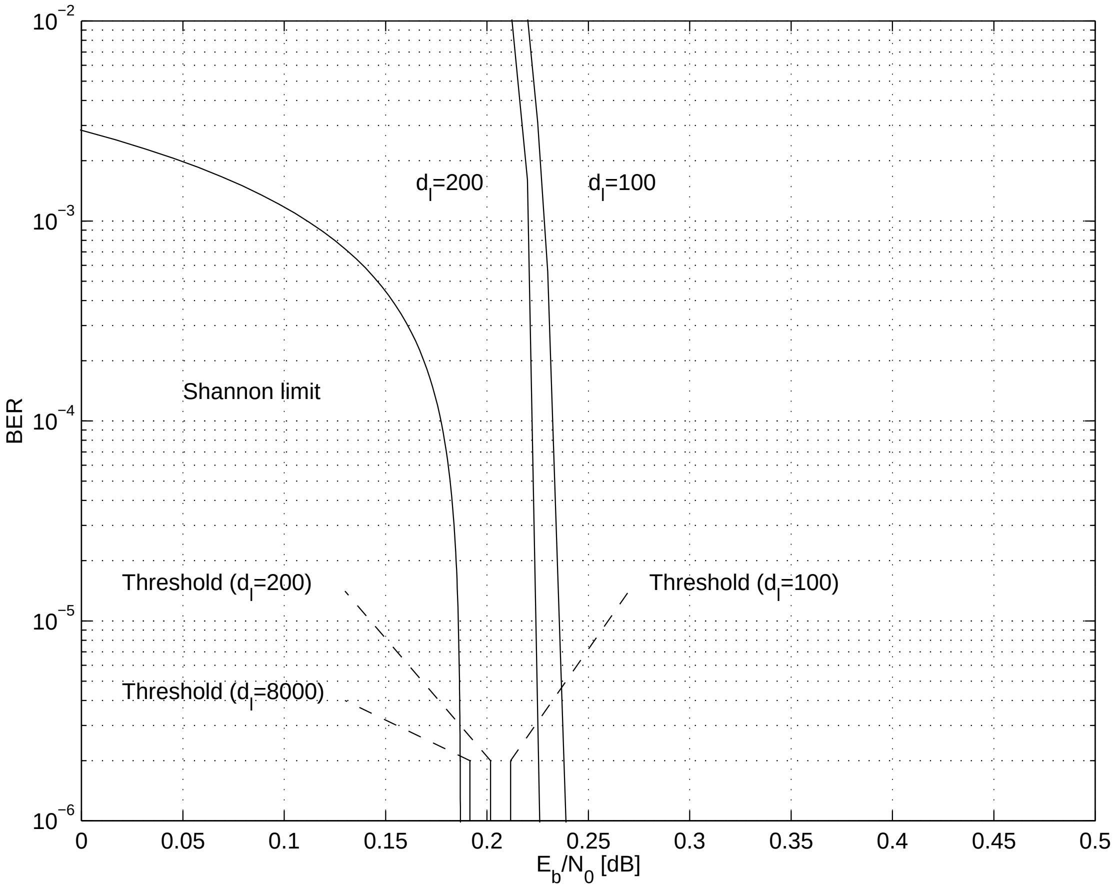

Figure 11. Asymptotic analysis with maximum left degree $d_l = 100, 200, 8000$ and simulations with $d_l = 100, 200$ and $n = 10^7$ of optimized rate-$\frac{1}{2}$ irregular LDPC codes [Chung *et al.*, 2001].

13.5.4 EXIT charts

For large LDPC codes, an efficient and quite accurate heuristic method of simulating sum-product decoding performance is to replace the probability distribution of the APP parameter p by a single summary statistic: most often, the conditional entropy $H(X \mid Y) = \mathsf{E}_{Y|0}[H(X \mid y)]$, or equivalently the mutual information $I(X;Y) = 1 - H(X \mid Y)$. We have seen that on the BEC, where $H(X \mid Y) = p$, this approach yields an exact analysis.

Empirical evidence shows that the relations between $H(X \mid Y)^{\text{out}}$ and $H(X \mid Y)^{\text{in}}$ for repetition and zero-sum nodes are very similar for all SBICs. This implies that degree distributions $\lambda(z), \rho(z)$ designed for the BEC may be expected to continue to perform well for other SBICs.

Similar EXIT chart analyses may be performed for turbo codes, RA codes, or for any codes in which there are left codes and right codes whose variables are shared through a large pseudo-random interleaver. In these cases the two relations between $H(X \mid Y)^{\text{out}}$ and $H(X \mid Y)^{\text{in}}$ often cannot be determined by analysis, but rather are measured empirically during simulations. Again, design is done by finding distributions of left and right codes such that the two resulting curves approach each other closely, without crossing. All of these classes of codes have now been optimized to approach capacity closely.